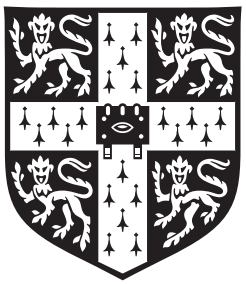
Improving Deep Learning with Probabilistic Approaches

James Urquhart Allingham
Darwin College



This dissertation is submitted for the degree of
Doctor of Philosophy

February 2024

To my brother, Taliesin,
without whom this would never have happened.

Declaration

This thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the preface and specified in the text. It is not substantially the same as any work that has already been submitted, or, is being concurrently submitted, for any degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the preface and specified in the text. It does not exceed the prescribed word limit for the relevant Degree Committee.

James Urquhart Allingham
February 2024

Acknowledgements

This thesis is the result of the support, encouragement, guidance, collaboration, and friendship of many people. There are certainly too many people to thank individually, but I will try to do so anyway.

Firstly, I must thank Javier Antorán. Without him, none of this work would have been possible. From the very beginning of the PhD, Javi has been a great source of inspiration and friendship. I have learnt a lot—both about machine learning and in general—from him. Even on projects that we did not work directly together, his impact can be seen. I am very grateful to have been assigned the seat behind him at the start of this journey, that he suggested we write a short workshop paper together, and that this led to a fruitful collaboration and friendship.

I must also thank my supervisor José Miguel Hernández-Lobato for his guidance, support, and encouragement throughout my PhD. Miguel has given me the freedom to pursue my own research interests and has always been available to discuss ideas and provide feedback. He has always had my best interests at heart. On the topic of supervisors and advisors, several others need to be thanked. Eric Nalisnick, my ELLIS co-supervisor, for his guidance and for providing me an opportunity to visit him in Amsterdam. I've thoroughly enjoyed our collaboration. Carl Rasmussen, my advisor, for his support and for always being available for an early morning run around Cambridge. And, Christian Steinruecken, who supervised my MPhil (Allingham, 2018), for encouraging me to pursue a PhD, and for putting me in a position to do so thanks to his generosity during (and after) the MPhil. A big thanks to Carl Rasmussen and Mark van der Wilk for examining this thesis—I thoroughly enjoyed the viva, and the thesis is certainly better for their input.

Rodolphe Jennaton, who hosted me for my first internship at Google Brain. Rodolphe was an incredible mentor and showed me what a joy a good manager-managee relationship can be. Even though Covid forced us to work remotely, working with Rodolphe was always a pleasure, and I would suggest everyone try it at least once! Jie Ren, who hosted me for my second internship at Google Brain, was also amazing. Jie always believed in me and my abilities, and was great fun to work with.

Collaboration has been the highlight of my PhD and is for me the most enjoyable part of research. I've been lucky to have worked with a huge number of fantastic people from whom I've learned a lot. They are, in approximate order of appearance, Javier Antorán, José Miguel Hernández-Lobato, Erik Daxberger, Eric Nalisnick, Florian Wenzel, Zelda Mariet, Basil Mustafa, Joan Puigcerver, Neil Houlsby, Vincent Fortuin, Balaji Lakshminarayanan, Jasper Snoek, Dustin Tran, Carlos Riquelme Ruiz, Rodolphe Jenatton, Mark Collier, Jeremiah Liu, Effrosyni Kokiopoulou, Chelsea Murray, David Janz, Riccardo Barbano, Shreyas Padhy, Metod Jazbec, Dan Zhang, Jie Ren, Michael Dusenberry, Xiuye Gu, Yin Cui, Bruno Mlodozeniec, David Krueger, and Richard Turner.

The Deep Learning Indaba community has been a huge source of support and inspiration. I would especially like to thank Ulrich Paquet, Avishkar Bhoopchand, Stephan Gouws, and Benji Rosman for their friendship and their generosity. They were always available to help me succeed in any way they could.

The Computational and Biological Learning group in Cambridge has provided a wonderful collaborative environment for research. I want to thank all of the members of the group, but especially Austin Tripp, Erik Daxberger, Talay Cheema, Jonny So, Valerii Likhosherstov, Marine Schimel, Javier Antorán, Kris Jensen, and Umang Bhatt, who started the PhD journey with me. My almost 6-month visit to AMLab at the University of Amsterdam was a highlight of my PhD, and I would like to thank everyone there for the fun times and the great discussions. Darwin College has been a wonderful home in Cambridge where I've thoroughly enjoyed the warm, relaxed, and friendly atmosphere.

I've had several teachers who have had a huge impact on my life. I would like to thank Bev McQueen, the late Louis Pasques, Ginny de Villiers, Andre de Wilzem, Barbara Heron, and Scott Hazelhurst for their support and encouragement. I would also like to thank Peter Loudon for his generous guidance and support.

This PhD would not have been possible without the financial support of the EPSRC, the Michael E. Fisher Studentship in Machine Learning, and the Qualcomm Innovation Fellowship. I am very grateful for their support.

Last, but certainly not least, I would like to thank my family and friends. My parents, Rob, Trish, and Richard who have always supported me. My sister Maria, who has lovingly kept me in check. My partner, Mimy who has been understanding, supportive, motivating, and, when needed, happy to help me escape the madness of the PhD. Carl, Paul, and Julian, whose friendships I will always cherish, and have all had an impact on my academic journey. Marine and Sonia, who helped keep me sane during Covid. And of course, Tali, who was instrumental in the journey that brought me here. I love you all.

Abstract

Despite its successes in scaling to real-world problems, deep learning is not without flaws. In particular, it struggles with uncertainty quantification and data efficiency. Probabilistic methods, while currently somewhat underappreciated by the wider machine learning community, provide calibrated uncertainty estimates and tend to shine in the low-data regime. It would seem that probabilistic methods complement deep learning. Thus, we ask the question, "How can probabilistic approaches be used to improve deep learning?"

On the topic of uncertainty estimation, we have three sets of contributions. Firstly, we show that probabilistic inference over the depth of a neural network not only side-steps challenges involved with scaling inference to the large weight spaces of modern neural networks but also provides well-calibrated uncertainty estimates and robust predictions. Furthermore, we leverage the uncertainty over depth for applications such as neural architecture search and active learning.

Secondly, we develop an alternative framework for dealing with these large weight spaces. We perform inference over a small subset of the weights in a neural network. We show that using crude approximations to select the subset of the weights is not very harmful compared to using them for inference over the weights. In particular, we find that capturing correlations between weights is essential for uncertainty estimation.

Thirdly, we investigate uncertainty estimation in sparse Mixture-of-Experts models. These models learn multiple diverse explanations of the data. We show that averaging these explanations results in robust predictions with well-calibrated uncertainty estimates. We provide an algorithm for doing so without incurring a heavy computational cost.

Finally, on the topic of data efficiency in deep generative models, we develop a generative model that learns which symmetry transformations are present in a dataset. This symmetry-aware generative model can be used to imbue standard deep generative models with inductive biases about the underlying generative process of the data. We experimentally show that this improves data efficiency.

Table of contents

List of figures

List of tables

List of terms

E^3 Efficient Ensemble of Experts 3, 78, 85–90, 92, 93, 173–182, 184, 186–192, 200, 201, 204

AUC Area Under Curve 72

BALD Bayesian Active Learning by Disagreement 48, 160, 161

BE Batch Ensemble 8, 9, 13, 81, 82, 85, 86, 89, 95, 174, 179–182, 186, 187

BMA Bayesian Model Averaging 27, 50, 154

BNN Bayesian Neural Network 7, 9–14, 21, 49, 50, 54, 55, 58, 95, 119, 121

BO Bayesian Optimisation 153, 154

BOHB Bayesian Optimisation and Hyperband 153, 154, 156

CNN Convolutional Neural Network 13, 28, 104, 106, 113, 118, 151, 221

DDN Deterministic Depth Network 44–47

DUN Depth Uncertainty Network 3, 25–27, 31, 33–35, 37, 40, 41, 43, 44, 46, 48–51, 75, 118, 119, 151–154, 156, 158, 160, 161

EBM Energy-based Model 17, 18, 23

ECE Expected Calibration Error 16, 17, 35, 36, 68, 70, 86, 88, 89, 92, 93, 180, 185, 186, 189, 190

ELBO Evidence Lower BOund 11, 22, 30–32, 44, 101, 153, 154, 209–211

FLOPs Floating Point Operations Per Second 77, 78, 82, 84, 86, 177, 180, 181, 204

GAN Generative Adversarial Network 18, 19, 23, 113, 223, 224

GGN Generalized Gauss-Newton 58, 60, 62–64, 66, 164, 165

GLM Generalized Linear Model 58

GP Gaussian Process 12, 33–35, 163

HMC Hamiltonian Monte Carlo 11–13, 49, 73, 119

HPO Hyperparameter Optimisation 153, 155, 156

HSV Hue Saturation Value 222, 225

IWAE Importance Weighted Autoencoder 22

IWLB Importance Weighted Lower Bound 111, 112

KL Kullback-Leibler 81, 87, 89, 181

KLD Kullback-Leibler Divergence 11, 22, 41, 43, 50, 61, 86

LDN Learnt Depth Network 44–47

LL Log Likelihood 35, 37, 40, 42, 43, 58, 65, 66, 68–70

LLM Large Language Model 121

MAP Maximum A Posteriori 10, 12, 53, 57, 59, 60, 64, 66, 67, 69, 163–165

MC Monte Carlo 9, 11–15, 30, 31, 49, 50, 67, 89, 95, 153, 154, 156, 157, 166, 180–182

MCDO Monte Carlo Dropout 48, 49, 160

MCMC Markov Chain Monte Carlo 10–12, 17

MFVI Mean-Field Variational Inference 31, 33, 37, 48, 49, 63, 69, 153, 154, 156, 157, 160

MIMO Multi-input Multi-output 8, 9, 90, 118, 174, 180–184, 186, 187

MLE Maximum Likelihood Estimation 13, 104, 210–212

MLL Marginal Log Likelihood 25, 28–33, 97, 115, 153, 209

MLP Multi-Layer Perceptron 35, 80, 171, 174, 200, 206, 207, 214, 215, 220, 221

MoE Mixture of Expert 77–86, 88, 94, 95, 120, 174, 175, 177, 179, 180, 184, 185, 200, 206, 207

MSA Multi-headed Self-Attention 80

NAS Neural Architecture Search 14, 25, 43

NF Normalising Flow 7, 19–21, 23, 102, 105, 120, 215

NLL Negative Log Likelihood 10, 16, 48, 49, 84, 86, 88, 89, 92, 93, 95, 154, 160, 161, 177, 178, 180, 185, 186, 189–191, 200, 201

NLP Natural Language Processing 80, 120

NN Neural Network 7–19, 23, 25–31, 35, 50, 51, 53–59, 61, 64–67, 69, 73–75, 77–79, 95, 101, 104, 111, 113, 114, 117–121, 152, 163, 164, 213, 214

NSF Neural Spline Flow 21, 215

OOD Out-of-distribution 18, 27, 31, 35, 36, 41, 42, 71, 72, 83, 92, 93, 95, 158, 189, 190

RCE Regression Calibration Error 35, 36

ReLU Rectified Linear Unit 64, 67, 152, 159, 160, 163, 164, 224

ResNet Residual Network 14, 31, 40, 41, 50, 64, 67, 69–71, 95, 121, 151, 157, 165, 166, 183

RMSE Root Mean Squared Error 35, 37

ROC Receiver Operating Characteristic 71, 72

SGD Stochastic Gradient Descent 11, 13, 31, 37, 41, 42, 50, 74, 152, 154, 157, 163–165, 170

SGHMC Stochastic Gradient HMC 11, 12

SGLD Stochastic Gradient Langevin Dynamics 11

SGM Symmetry-aware Generative Model 97, 99, 100, 103, 105–108, 111, 112, 115, 120, 121, 211, 223–226, 230

SSL Self-Supervised Learning 102, 104, 105, 209, 210, 212, 213, 216, 217

SWAG Stochastic Weight Averaging Gaussian 63, 67, 69, 72, 165, 166

TCE Tail Calibration Error 35–37

V-MoE Vision-MoE 80–84, 86, 87, 89, 90, 92–95, 118, 170, 173–183, 185–187, 189–191, 200, 201, 203, 204

VAE Variational Autoencoder 7, 19–23, 97, 102, 111, 112, 114, 120, 121, 209–211, 215, 216, 221, 225

VI Variational Inference 10, 12–14, 19, 21, 22, 25, 29–33, 50, 95, 119, 154

ViT Vision Transformer 80, 82–84, 90, 92–95, 169, 172–175, 177, 178, 180, 181, 183, 184, 186–190, 200, 201, 204, 206, 207

VOGN Variational Online Gauss-Newton 67, 69, 165, 166

List of symbols

Variables

$a,\ x,\ \theta,\ \phi,$ etc.	scalars
$\mathrm{a},\ \mathrm{x},\ \theta,\ \phi$	random scalars
$\boldsymbol{a},\ \boldsymbol{x},\ \boldsymbol{\theta},\ \boldsymbol{\phi}$	vectors
$\mathbf{a},\ \mathbf{x},\ \boldsymbol{\theta},\ \boldsymbol{\phi}$	random vectors
$\boldsymbol{A},\ \boldsymbol{X}$	matrices
$\mathbf{A},\ \mathbf{X}$	random matrices

Probability

$p(\mathrm{a})$	a PDF or PMF for the random variable a
$q(\mathrm{a})$	another PDF or PMF for the random variable a
$p(\mathrm{a} = a),\ p(a)$	a density or mass for the event $\mathrm{a} = a$
$p(\mathrm{y},\ \mathrm{x})$	a joint probability for y and x
$p(\mathrm{y} \mid \mathrm{x})$	a conditional probability for y given x
$p_{\boldsymbol{\phi}}(\cdot)$	a probability with parameters $\boldsymbol{\phi}$
$\mathcal{N}(\mathrm{x} \mid \mu,\ \sigma^2)$	a Normal distribution with mean μ and variance σ^2
$\mathbb{E}_{p(\mathrm{x})}\left[f(\mathrm{x})\right]$	the expected value $\int f(\mathrm{x}) p(\mathrm{x}) \mathrm{dx}$
$\mathbb{H}\left[p(\mathrm{x})\right]$	the entropy of $p(\mathrm{x})$
$\mathbb{D}_{\mathrm{KL}}\left[q(\mathrm{x}) \,||\, p(\mathrm{x})\right]$	the KL divergence $\mathbb{E}_{q(\mathrm{x})}\left[\log \frac{q(\mathrm{x})}{p(\mathrm{x})}\right]$ between $q(\mathrm{x})$ and $p(\mathrm{x})$

Miscellaneous

$\mathcal{D}$	a dataset
$f_1 \circ f_2$	the composition of the functions f_1 and f_2
$\mathbb{1}[\texttt{cond}]$	the indicator function; returns 1 if `cond` is True otherwise 0

Chapter 1

Introduction

Deep learning (LeCun et al., 2015) is an extremely successful machine learning paradigm that has been driving renewed excitement around machine learning and artificial intelligence since AlexNet (Krizhevsky et al., 2012) won the ImageNet LSVRC competition (Deng et al., 2009; Russakovsky et al., 2015). Deep learning is not only responsible for excitement around ML but also a large number of commercial and research successes:

- Superhuman performance in image classification (He et al., 2016a),
- World-wide deployment of human-level machine translation (Wu et al., 2016),
- Superhuman performance in playing challenging games such as Go (Silver et al., 2016), StarCraft II (Vinyals et al., 2019), and Dota 2 (Berner et al., 2019),
- Dermatologist-level classification of skin cancer (Esteva et al., 2017), and
- State-of-the-art protein folding predictions (Senior et al., 2020).

Most recently, deep learning has been responsible for a new wave of excitement surrounding artificial intelligence with the development of strikingly human-like chatbots such as ChatGPT—built upon GPT3/4 (Brown et al., 2020; OpenAI, 2023)—and Gemini (Gemini Team, 2023), as well as text-to-image (and video) systems like DALLE-2 (Ramesh et al., 2022) and Imagen (Saharia et al., 2022).

However, despite the successes of deep learning, it is not without its flaws. One such flaw is that deep learning models are bad at *knowing when they don't know*. That is, they are unable to robustly and reliably quantify the uncertainty in their predictions. To further illustrate this problem, consider self-driving cars. This is a safety-critical application for which deep learning is already being used in production (e.g., Tesla

Autopilot (Tesla, 2024)). Self-driving car technology must be able to handle never-before-encountered traffic situations since it is impossible to collect training data involving all possible combinations of vehicles, pedestrians, weather conditions, traffic signs, and other objects/events. When these situations are encountered—and the software cannot possibly act in a safe and reliable manner—it makes sense for the behaviour of the vehicle to be modified (e.g., slowing down to a stop on the side of the road). However, such fallbacks can only be enabled if the software is aware that it is currently acting outside of its normal operating conditions. Without the ability to *know what it doesn't know*, this situation cannot be detected. Other applications that require reliable uncertainty estimates include exploration in reinforcement learning (Osband et al., 2016), active learning (Settles, 2009), optimally combining the predictions of different models, and bet sizing using the Kelly criterion (Kelly, 1956).

Another flaw is that most deep learning models tend to be discriminative rather than generative and typically do not include strong inductive biases about the underlying causal generative process for the data. As a result, deep learning models tend to be data hungry and can struggle with generalisation to out-of-distribution domains. Furthermore, the ability of a model to generate data is useful in its own right and can help us *debug* pathologies of our model (Ghahramani, 2015). Consider a situation in which the training data is not representative of the population to which the model is being applied. This problem can be diagnosed by generating samples from the model and observing the disparity between the model's view of the world and reality.

Probabilistic machine learning (Ghahramani, 2015), in contrast to deep learning, does not suffer from these flaws. Uncertainty plays a fundamental part in the probabilistic framework, and there is an intimate link between discriminative and generative models via *Bayes' rule*—a discriminative model $p(y \mid x)$, which predicts a label y given an example x, can be inverted to give a generative model $p(x \mid y)$ which can create new examples given a label, and vice versa. Probabilistic methods also offer other advantages such as principled model comparison (MacKay, 2003), and natural handling of missing data via latent variable inference—e.g., with the EM algorithm (Ghahramani and Jordan, 1993)—to name a few.

Ideally, we want models which leverage both the raw predictive power of deep learning and the complementary advantages of the probabilistic framework. However, combining the probabilistic approach with deep learning provides many challenges. This thesis aims to address some of these challenges by developing new methods for combining deep learning and probabilistic modelling to improve (1) the uncertainty estimates and (2) the generative modelling capabilities of deep neural networks.

1.1 Thesis Outline and Contributions

This thesis is structured as follows.

Chapter 2 provides an overview of the probabilistic framework and Bayesian inference applied to deep learning. It covers ensembles (Section 2.1.1), Bayesian neural networks (Section 2.1.2), computation and evaluation of uncertainty estimates (Sections 2.1.3 and 2.1.4), and deep generative models (Section 2.2) with a focus on variational autoencoders and normalising flows.

Chapter 3 explores the use of probabilistic inference for determining the depth of a neural network, and how the uncertainty over the depth can be translated to uncertainty over the predictions of the model. This chapter is primarily based on "Depth Uncertainty in Neural Networks" (Antorán et al., 2020) with additional content from "Depth Uncertainty Networks for Active Learning" (Murray et al., 2021b). The main contribution is the development of a *Depth Uncertainty Network (DUN)* which demonstrates that uncertainty over the depth leads to to well-calibrated predictive uncertainty estimates. Minor contributions include the application of DUNs to neural architecture search and active learning.

Chapter 4 focuses on probabilistic inference for neural network weights. In particular, the chapter addresses the challenge of applying expressive Bayesian inference to deep neural networks with many parameters. This chapter is based on "Bayesian Deep Learning via Subnetwork Inference" (Daxberger et al., 2021b). The main contribution is the development of *subnetwork inference*—a framework for scalable Bayesian inference in the large parameter spaces of deep neural networks. When applied to Laplace approximations, subnetwork inference provides calibrated uncertainty estimates that are competitive with state-of-the-art methods.

Chapter 5 investigates uncertainty estimation for modern neural networks—such as Transformers (Vaswani et al., 2017) and Sparse MoEs (Shazeer et al., 2017)—for which the posterior inference-based approaches are not applicable due to the extremely large number of parameters. Instead, taking inspiration from Bayesian model averaging, different explanations of the data are averaged to obtain robust predictions with calibrated uncertainty estimates. This chapter is based on "Sparse MoEs meet Efficient Ensembles" (Allingham et al., 2022b). There are two main contributions: (1) a demonstration that Sparse MoEs and ensemble methods (which share several similarities) can be combined to obtain improved predictive performance and better uncertainty estimates, and (2) an *Efficient Ensemble of Experts (E^3)* model which improves the uncertainty estimates of Sparse MoEs without sacrificing computational efficiency.

Chapter 6 tackles the problem of data efficiency in deep generative models. Concretely, the chapter focuses on learning symmetries in the data in an unsupervised manner and incorporating these symmetries into "black-box" generative models without strong inductive biases to improve their data efficiency. This chapter is based on "A Generative Model of Symmetry Transformations" (Allingham et al., 2024). The main contributions are the development of an algorithm for learning symmetries in the data and a demonstration that incorporating these symmetries into generative models improves their data efficiency.

Finally, Chapter 7 concludes the thesis and discusses future directions for research.

List of Papers

This thesis is based on several papers which have been written throughout my PhD with many fantastic collaborators. For the sake of completeness, a list of these papers is included below. I have also included several papers that I have contributed to but are not included in this thesis. The papers included in the thesis are indicated with an **bold** titles. Equal contributions are indicated with an asterisk (*).

Peer-reviewed Conference and Journal Papers

James Urquhart Allingham*, Javier Antorán*, and José Miguel Hernández-Lobato. "**Depth Uncertainty in Neural Networks.**" In *Advances in Neural Information Processing Systems 33, NeurIPS*, 2020.

Erik Daxberger, **James Urquhart Allingham***, Eric Nalisnick*, Javier Antorán*, and José Miguel Hernández-Lobato. "**Bayesian Deep Learning via Subnetwork Inference.**" In *Proceedings of the 38th International Conference on Machine Learning, ICML*, 2021.

Chelsea Murray, **James Urquhart Allingham**, Javier Antorán, and José Miguel Hernández-Lobato. "Addressing Bias in Active Learning with Depth Uncertainty Networks... or Not." In *Proceedings of I (Still) Can't Believe It's Not Better! Workshop at NeurIPS*, 2021.

James Urquhart Allingham, Florian Wenzel, Zelda E. Mariet, Basil Mustafa, Joan Puigcerver, Neil Houlsby, Ghassen Jerfel, Vincent Fortuin, Balaji Lakshminarayanan, Jasper Snoek, Dustin Tran, Carlos Riquelme Ruiz, and Rodolphe Jenatton. "**Sparse MoEs meet Efficient Ensembles.**" *Transactions on Machine Learning Research*, 2022.

Vincent Fortuin, Mark Collier, Florian Wenzel, **James Urquhart Allingham**, Jeremiah Liu, Dustin Tran, Balaji Lakshminarayanan, Jesse Berent, Rodolphe Jenatton, and Effrosyni Kokiopoulou. "Deep Classifiers with Label Noise Modelling and Distance Awareness." *Transactions on Machine Learning Research*, 2022.

Javier Antorán, **James Urquhart Allingham***, David Janz*, Erik Daxberger, Riccardo Barbano, Eric Nalisnick, and José Miguel Hernández-Lobato. "Adapting the Linearised Laplace Model Evidence for Modern Deep Learning." In *Proceedings of the 39th International Conference on Machine Learning, ICML*, 2022.

James Urquhart Allingham*, Jie Ren*, Michael W. Dusenberry, Xiuye Gu, Yin Cui, Dustin Tran, Jeremiah Zhe Liu, and Balaji Lakshminarayanan. "A Simple Zero-shot Prompt Weighting Technique to Improve Prompt Ensembling in Text-Image Models." In *Proceedings of the 40th International Conference on Machine Learning, ICML*, 2023.

Metod Jazbec, **James Urquhart Allingham**, Dan Zhang, and Eric Nalisnick. "Towards Anytime Classification in Early-Exit Architectures by Enforcing Conditional Monotonicity." In *Advances in Neural Information Processing Systems 37, NeurIPS*, 2023.

James Urquhart Allingham, Bruno Mlodozeniec, Shreyas Padhy, Javier Antorán, David Krueger, Richard Turner, Eric Nalisnick, and José Miguel Hernández-Lobato. "**A Generative Model of Symmetry Transformations.**" In *Proceedings of the 41st International Conference on Machine Learning, ICML*, 2024.

Peer-reviewed Workshop and Symposium Papers

James Urquhart Allingham*, Javier Antorán*, and José Miguel Hernández-Lobato. "Variational Depth Search in ResNets." *1st ICLR Workshop on Neural Architecture Search*, 2020.

Chelsea Murray, **James Urquhart Allingham**, Javier Antorán, and José Miguel Hernández-Lobato. "**Depth Uncertainty Networks for Active Learning.**" *NeurIPS Workshop on Bayesian Deep Learning*, 2021.

Javier Antorán, **James Urquhart Allingham**, David Janz, Erik Daxberger, Eric Nalisnick, and José Miguel Hernández-Lobato. "Linearised Laplace Inference in Networks with Normalisation Layers and the Neural g-Prior." In *Fourth Symposium on Advances in Approximate Bayesian Inference, AABI*, 2022.

James Urquhart Allingham and Eric Nalisnick. "A Product of Experts Approach to Early-Exit Ensembles." *1st ICML Workshop on Dynamic Neural Networks*, 2022.

James Urquhart Allingham, Javier Antoran, Shreyas Padhy, Eric Nalisnick, and José Miguel Hernández-Lobato. "Learning Generative Models with Invariance to Symmetries." *NeurIPS Workshop on Symmetry and Geometry in Neural Representations*, 2022.

Chapter 2

Background

This chapter covers two applications of probabilistic inference in deep learning: uncertainty estimation (Section 2.1) and deep generative modelling (Section 2.2). Section 2.1 first covers two popular approaches to uncertainty estimation in deep learning: ensembles (Section 2.1.1) and Bayesian Neural Networks (BNNs) (Section 2.1.2). Then, Section 2.1.3 covers how to compute uncertainties for NNs and Section 2.1.4 covers how to evaluate the quality of those uncertainty estimates[1]. Section 2.1 will be important for Chapters 3, 4 and 5, which all centre around methods for uncertainty estimation in deep learning. Section 2.2 introduces deep generative modelling and discusses several popular approaches with a particular focus on Variational Autoencoders (VAEs) (Section 2.2.2) and Normalising Flows (NFs) (Section 2.2.1) Section 2.2 will be required for Chapter 6, which focuses on improving the data-efficiency of deep generative models and builds on NFs and VAEs.

2.1 Uncertainty Estimation in Deep Learning

Uncertainty estimation for NNs is a well-studied problem. The two most common approaches are *ensemble methods* and *BNNs*. Ensembles train multiple independent NNs and aggregate their predictions to obtain uncertainty estimates. BNNs use Bayesian inference to obtain a posterior distribution over the parameters of a NN, which is then used to obtain predictive uncertainty estimates. In this section, we review both approaches.

[1]These two subsections closely follow Appendix C of Antorán et al. (2020).

2.1.1 Ensembles

Ensembles have long been used to improve the predictive performance of machine learning models (Hansen and Salamon, 1990; Geman et al., 1992; Krogh and Vedelsby, 1995; Opitz and Maclin, 1999; Dietterich, 2000). Lakshminarayanan et al. (2017) propose deep ensembles—a simple but effective method for obtaining uncertainty estimates from NNs—that trains multiple independent networks and aggregates their predictions. The uncertainty in the predictions of the ensemble is then quantified using the disagreement between the predictions of the ensemble members. While this method provides very strong results, both in terms of accuracy and uncertainty quantification, it is limited by its computational cost. Nonetheless, deep ensembles remain a very strong baseline for uncertainty estimation in NNs.

Several works (Huang et al., 2017; Garipov et al., 2018; Maddox et al., 2019) reduce the cost of training an ensemble at the price of reduced predictive performance (Ashukha et al., 2020). Two of the most popular of these *efficient ensemble* methods are Batch Ensembles (BEs) (Wen et al., 2020) and Multi-input Multi-output (MIMO) ensembles (Havasi et al., 2020).

Batch Ensembles. BEs, proposed by Wen et al. (2020), train an ensemble of NNs in which some "slow" parameters are shared across the ensemble members, while the "fast" parameters are unique to each ensemble member. Concretely, the two per-ensemble vectors $\mathbf{r}_m \in \mathbb{R}^D$ and $\mathbf{s}_m \in \mathbb{R}^L$ are combined with a shared weight matrix $\mathbf{U} \in \mathbb{R}^{D \times L}$, to create the m^{th} member's weight matrix

$$\mathbf{U}_m = \mathbf{U} \cdot (\mathbf{r}_m \mathbf{s}_m^\intercal), \tag{2.1}$$

where $\cdot$ is elementwise multiplication. A BE can be trained in a single forward and backward pass by tiling the input batch and vectorising (2.1). Thus, the computational cost of a BE is approximately the same as a standard NN, assuming that there is enough memory to hold the tiled batch. (Wen et al., 2020) show that the success of BEs relative to other efficient ensemble methods is due to their more diverse predictions.

MIMO Ensembles. MIMO ensembles, proposed by Havasi et al. (2020), train a single NNs to make multiple predictions. A standard NN is augmented such that there are M input and output layers, where M is the ensemble size. Each input layer is fed a different input, and each output layer is trained to predict the corresponding target. At test time, each input layer is fed the same input, and the predictions of the output

layers are aggregated to obtain a single prediction. Despite only having a small number of member-specific parameters (the input and output layers), MIMO ensembles are competitive with BEs and standard ensembles in several settings. On the other hand, for large values of M, MIMO ensemble performance can degrade if the base NN is not sufficiently over-parameterised with respect to the size of the dataset. As with any ensemble method, sufficient over-parameterisation is required to learn multiple diverse explanations of the data. However, MIMO ensembles are particularly sensitive to this issue because, except for the input and output layers, the number of parameters does not scale with M. While BEs are also somewhat sensitive to this issue, they are more resilient because fast parameters can be added to each layer of the NN.

2.1.2 Bayesian Neural Networks

A natural solution to the problem of overconfidence in deep networks is the Bayesian approach, which uses Bayes' rule to calculate the posterior distribution over the parameters of a NN given the likelihood—the probability of observing the data given the parameters—and some prior belief about the parameters:

$$p(\boldsymbol{\theta} \mid \mathcal{D}, \mathcal{M}) = \frac{p(\mathcal{D} \mid \boldsymbol{\theta}, \mathcal{M}) p(\boldsymbol{\theta}, \mathcal{M})}{\int p(\mathcal{D} \mid \boldsymbol{\theta}, \mathcal{M}) p(\boldsymbol{\theta}, \mathcal{M}) \mathrm{d}\boldsymbol{\theta}}, \tag{2.2}$$

where $\boldsymbol{\theta}$ are the parameters of the NN, $\mathcal{D}$ is the observed data, and $\mathcal{M}$ represents our modelling assumptions such as the number of layers and the choice of activation function in the NN. The other term in (2.2), also written as $p(\mathcal{D} \mid \mathcal{M})$, is the evidence a.k.a. the marginal likelihood. It can be interpreted as the probability of the data given our modelling choices. For the sake of conciseness, in the rest of this thesis, we will drop the explicit dependence on $\mathcal{M}$. Once the posterior has been calculated, predictions for a new test point $\mathbf{x}^*$ can be made by marginalising the parameters:

$$p(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D}) = \int p(\mathbf{y}^* \mid \mathbf{x}^*, \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid \mathcal{D}) \mathrm{d}\boldsymbol{\theta}. \tag{2.3}$$

Unfortunately, the integrals in Equations (2.2) and (2.3) are intractable to compute for NNs. As a result, BNNs require approximations in practice. For (2.3), the solution is to approximate the integral using Monte Carlo (MC) sampling:

$$p(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D}) \approx \frac{1}{N} \sum_{n=1}^{N} p(\mathbf{y}^* \mid \mathbf{x}^*, \boldsymbol{\theta}_n), \quad \boldsymbol{\theta}_n \sim p(\boldsymbol{\theta} \mid \mathcal{D}). \tag{2.4}$$

In the case of the evidence in (2.2), approximations generally come in two forms.

1. Simplifying assumptions about the form of the posterior distribution, which allow us to either avoid or simplify the calculation of the evidence.

2. Using Markov Chain Monte Carlo (MCMC) methods to sample from the posterior, without directly calculating the evidence.

Some of the earliest work on BNNs—falling into the first category—was done by MacKay (1992), who proposed a Laplace approximation of the posterior. The Laplace approximation assumes the posterior is a unimodal Gaussian

$$p(\theta \mid \mathcal{D}) = \mathcal{N}(\theta \mid \boldsymbol{\theta}^*, \Sigma), \tag{2.5}$$

where $\boldsymbol{\theta}^*$ are the MAP parameters found using standard gradient descent on the regularised Negative Log Likelihood (NLL) loss, and Σ is the inverse Hessian of the negative log posterior, which is equivalent to the Hessian of the loss evaluated at $\boldsymbol{\theta}^*$:

$$\Sigma^{-1} = \nabla^2_\theta \left(-\log p(\mathcal{D} \mid \theta) - \log p(\theta)\right)\Big|_{\theta=\boldsymbol{\theta}^*}. \tag{2.6}$$

However, computing the Hessian for a modern NN—with on the order of 10^6 parameters—would require on the order of 10^{12} bytes of memory and a non-trivial amount of compute. To overcome this limitation, MacKay (1992) assumes that the Hessian is diagonal, i.e., that all parameters are independent from one another. Ritter et al. (2018) relax the independence assumption using the Kronecker-factorised covariance matrix approximation of Martens and Grosse (2015). With this block-diagonal factorisation, it is only assumed that parameters in *different* layers are independent.

Other approximations include expectation propagation (Hernández-Lobato and Adams, 2015) and Variational Inference (VI) (Hinton and van Camp, 1993; Graves, 2011; Blundell et al., 2015). The VI approach was pioneered by Hinton and van Camp (1993), who require manual derivation of analytical expressions for the posterior, which limits applicability to all but the simplest NNs. Graves (2011) built on this work, allowing VI to automatically be applied to almost any neural network at the cost of biased gradients and the restriction that the posterior for each weight be Gaussian. Blundell et al. (2015) further built on this work by incorporating ideas from stochastic variational inference (Hoffman et al., 2013; Ranganath et al., 2014) to provide unbiased estimates of the gradients and lift the Gaussian restriction. At a high level, VI methods avoid the intractable integral in (2.2) by approximating $p(\theta \mid \mathcal{D})$ with $q_\phi(\theta)$, a family of distributions parameterised by

$\boldsymbol{\phi}$. The value of $\boldsymbol{\phi}$, and therefore the specific form of $q(\cdot)$, is chosen by minimising the Kullback-Leibler Divergence (KLD) between $p(\boldsymbol{\theta} \mid \mathcal{D})$ and $q_{\boldsymbol{\phi}}(\boldsymbol{\theta})$, using gradient descent:

$$\boldsymbol{\phi} = \arg\min_{\boldsymbol{\phi}} \mathbb{D}_{\mathrm{KL}}\left[q_{\boldsymbol{\phi}}(\boldsymbol{\theta}) \,||\, p(\boldsymbol{\theta} \mid \mathcal{D})\right] \tag{2.7}$$

$$= \arg\max_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{\theta})}\left[\log p(\mathcal{D} \mid \boldsymbol{\theta})\right] - \mathbb{D}_{\mathrm{KL}}\left[q_{\boldsymbol{\phi}}(\boldsymbol{\theta}) \,||\, p(\boldsymbol{\theta})\right] \tag{2.8}$$

$$= \arg\max_{\boldsymbol{\phi}} \mathcal{L}(\boldsymbol{\phi}). \tag{2.9}$$

$\mathcal{L}(\boldsymbol{\phi})$ is called the Evidence Lower BOund (ELBO). The two terms in the ELBO neatly trade off keeping the model simple (the KL term) and explaining the data (the likelihood term). Blundell et al. (2015) use MC sampling and the *reparameterisation trick* (Kingma and Welling, 2014) to calculate an unbiased estimate of the ELBO for any NN, which allows the variational parameters $\boldsymbol{\phi}$ to be learnt via backpropagation:

$$\mathcal{L}(\boldsymbol{\phi}) \approx \frac{1}{N}\sum_{n=1}^{N}\left[\log p(\mathcal{D} \mid \boldsymbol{\theta}_n) - \log q_{\boldsymbol{\phi}}(\boldsymbol{\theta}_n) + \log p(\boldsymbol{\theta}_n)\right], \quad \boldsymbol{\theta}_n \sim q_{\boldsymbol{\phi}}(\boldsymbol{\theta}). \tag{2.10}$$

For many choices of prior and variational family, the KL term in the ELBO can be calculated analytically. However, (2.10) is more general, and Blundell et al. (2015) did not find that using an analytical KLD improved or worsened performance. Finally, note that Hinton and van Camp (1993), Graves (2011), and Blundell et al. (2015) make a mean-field assumption, i.e., all parameters are independent from one another.

Some of the first work in the second category was done by (Neal, 1995) who uses Hamiltonian Monte Carlo (HMC) (Duane et al., 1987; Neal, 2011; Betancourt, 2017) to sample from the posterior. HMC solves a key problem with a naive application of MCMC methods to BNNs: random walk behaviour (Neal, 1995). Without the reduction in random walk behaviour provided by HMC, MCMC methods would be too slow to be of any practical use in BNNs. A major limitation of HMC is the need to compute the gradient of the potential energy function on the full dataset (Chen et al., 2014). One solution to this problem is Stochastic Gradient Langevin Dynamics (SGLD) (Welling and Teh, 2011), which is a mini-batch friendly learning algorithm which can be seen as *Stochastic Gradient Descent (SGD) with added noise*. In another light, SGLD can be seen as a variant of HMC with only one leapfrog iteration and without the important momentum term (whose absence results in increased random-walk behaviour). Another solution, more in line with HMC, is Stochastic Gradient HMC (SGHMC) (Chen et al., 2014), which counteracts the negative effects of using a noisy mini-batch estimate of

the potential energy's gradient, thereby allowing MCMC approaches to scale to large datasets. However, Betancourt (2015) shows that SGHMC is biased. Nonetheless, given very large compute resources, full-batch HMC outperforms MAP-estimated NNs, VI, deep ensembles, and other MCMC methods (Izmailov et al., 2021b). Although it is not clear how well HMC approximates the true posterior distributions of large NNs, its strong predictive performance and asymptotic guarantees mean that HMC—coupled with impractically large compute resources—arguably provides a useful point of reference for understanding Bayesian inference in NNs.

An alternative solution to the problem of approximating BNNs is based on the work of Neal (1995), who showed that, under certain conditions, a neural network with a single infinitely wide hidden layer is equivalent to a Gaussian Process (GP). The result of Neal (1995) has been extended to more general classes of neural networks, including deep and convolutional networks (Lee et al., 2018; de G. Matthews et al., 2018; Garriga-Alonso et al., 2019). Rather than performing approximate inference for a neural network, we can perform exact inference by working with its limiting GP. Unfortunately, naive inference in GPs is an $\mathcal{O}(N^3)$ operation in the dataset size, and while more efficient approximations exist (e.g., Gibbs and MacKay (1996); Snelson and Ghahramani (2005); Titsias (2009); Hensman et al. (2013); Gardner et al. (2018)), the scalability of this approach is still limited.

A recent approach to approximate inference in BNNs is to take advantage of Bayesian interpretations for the stochastic nature of standard NN training methods. This idea was introduced by Gal and Ghahramani (2016), who reinterpret binary dropout (Srivastava et al., 2014) as VI. This idea was extended to Gaussian dropout by Kingma et al. (2015). Teye et al. (2018) and Atanov et al. (2019a) applied the same idea to batch normalization (Ioffe and Szegedy, 2015), noting that many popular NN architectures no longer make use of dropout. Similarly, Khan et al. (2018) inject noise into the Adam optimiser (Kingma and Ba, 2015). These methods are particularly attractive because they require little to no changes in code and training procedure, and they are scalable to large datasets (e.g., Osawa et al. (2019) scale the method of Khan et al. (2018) to ImageNet). However, they are not without shortcomings. For example, Osband (2016) notes that the MC Dropout (Gal and Ghahramani, 2016) posterior predictive does not concentrate as more data are observed.

One of the challenges for BNNs is the high dimensionality of the parameter space. With tens of millions of parameters not being uncommon for modern NNs, many approaches for probabilistic inference perform poorly. A solution to this is subspace inference (Izmailov et al., 2019), which is motivated by the fact that the *intrinsic dimensionality* of a

modern NN can be orders of magnitude lower than the number of parameters (Li et al., 2018). Subspace inference is compatible with a variety of Bayesian inference methods such as HMC, VI, and the Laplace approximation. Additionally, because of the low dimensionality of the subspace, flexible variational families such as RealNVP (Dinh et al., 2017) can be used. Izmailov et al. (2019) demonstrate that subspace inference performs favourably compared to baselines such as MC Dropout and Kronecker-factorised Laplace approximations on small-scale tabular regression and image classification benchmarks. A recent approach related to subspace inference is BNNs with rank-1 factors (Dusenberry et al., 2020a), which builds on the BE method (Wen et al., 2020). This approach uses VI to learn the posterior distributions for the two rank-1 vectors $\mathbf{r}_m$ and $\mathbf{s}_m$ in (2.1), while learning $\mathbf{U}$ via MLE. This approach is competitive with both deterministic NN training and Deep Ensembles on ImageNet.

Another potential issue with the BNN approach is that it is not clear how to place reasonable priors over the weights of the network, particularly in high-dimensional weight spaces. Indeed, many of the BNN methods listed above assume independent Gaussian priors for each of the weights. Wenzel et al. (2020a) show that without up weighing the likelihood the performance of BNNs with mean-field Gaussian priors—even with accurate posterior inference via HMC—can be worse than standard SGD trained NNs. Fortuin et al. (2022b) support these findings by demonstrating that this *cold posterior* effect can be mitigated by using heavier-tailed prior distributions. They also show that spatial correlations in CNNs weight priors, especially in later layers, improve performance. Their findings highlight the importance of choosing good priors rather than defaulting to independent Gaussian distributions. On the other hand, Izmailov et al. (2021b) suggest that the cold-posterior effect is largely an artefact of data-augmentation. They also found that performance was largely robust to the choice of prior. In a follow-up paper, Izmailov et al. (2021a) demonstrate that performance under distribution shift is highly sensitive to the choice of prior. Finally, Foong et al. (2020) show that the *mean-field* prior used by VI and MC Dropout does not allow *in-between* uncertainty to be captured by 1-hidden layer NNs and that deeper networks struggle to capture it, despite theoretically being able to do so.

Recent work (Hafner et al., 2019; Sun et al., 2019; Ma et al., 2019) proposes functional inference as a solution to the challenge of choosing priors in weight-space. However, functional inference methods are hampered by various limitations (Nalisnick and Smyth, 2018; Pearce et al., 2019; Nalisnick et al., 2021; Burt et al., 2021), such as approximation of divergences for stochastic processes, and have yet to see widespread adoption. On the other hand, Wilson (2020) and Wilson and Izmailov (2020) argue that the strength of

the Bayesian inference for NNs does not lie in the choice of prior over weight—since the choice of NN architecture is essentially a choice of function space prior—but rather in the marginalisation over multiple explanations of the data that takes place when making predictions. In this light, even deep ensembles can be viewed as "Bayesian" in a loose sense. In fact, the strong performance of deep ensembles seems to stem from the fact that their predictions marginalise multi-modal explanations of the data (Fort et al., 2019).

Instead of performing inference over the weights of a NN, as in BNNs, one can perform inference over the structure of NNs. An early work in this direction is *automatic relevance detection* prior (MacKay, 1994), which can perform inference over NN width. Since then, several similar approaches have been introduced, such as Lawrence (2001b); Ghosh et al. (2019). Nalisnick et al. (2019a) performs inference over NN depth with an *automatic depth determination* prior. Perhaps unsurprisingly, probabilistic inference over NN structure has been applied in the context of Neural Architecture Search (NAS). For example, Dikov and Bayer (2019) learn the depth and width of a ResNet using VI.

2.1.3 Computing Uncertainties

In this section, we discuss how to compute uncertainties for NNs. Let us consider NNs, which parametrises two kinds of distributions: the categorical and the Gaussian, for classification and regression problems, respectively. Now, let us consider a random variable $\boldsymbol{\xi}$, with distribution $q(\boldsymbol{\xi})$, that induces stochasticity in our NNs. For example, in the case of BNNs, we would have a distribution over weights of the network. But, as we will see in Chapter 3, this could also be a distribution over the depth of the network.

For classification models, our networks output a vector—either of logits or probabilities—with elements $f_k(\mathbf{x}, \boldsymbol{\xi})$, corresponding to classes $\{c_k\}_{k=1}^K$. The likelihood function is $p(\mathrm{y} \mid \mathbf{x}, \boldsymbol{\xi}) = \mathrm{Cat}(\mathrm{y} \mid f(\mathbf{x}, \boldsymbol{\xi}))$. Through marginalisation, the uncertainty in $\boldsymbol{\xi}$ is translated into uncertainty in predictions. In some cases, this marginalisation can be computed exactly; however, in most cases, we resort to approximating it with MC sampling:

$$p(\mathrm{y}^* \mid \mathbf{x}^*, \mathcal{D}) = \mathbb{E}_{q(\boldsymbol{\xi})}\left[p(\mathrm{y}^* \mid \mathbf{x}^*, \boldsymbol{\xi})\right] \tag{2.11}$$

$$\approx \frac{1}{M}\sum_{m=0}^{M} \mathrm{Cat}(\mathrm{y} \mid f(\mathbf{x}^*, \boldsymbol{\xi}_m)); \quad \boldsymbol{\xi}_m \sim q(\boldsymbol{\xi}). \tag{2.12}$$

In both the exact and approximate cases, the resulting predictive distribution is categorical. We quantify its uncertainty using entropy:

$$\mathbb{H}\left[p\left(\mathrm{y}^* \mid \mathbf{x}^*,\, \mathcal{D}\right)\right] = \sum_{k=1}^{K} p\left(\mathrm{y}^* = c_k \mid \mathbf{x}^*,\, \mathcal{D}\right) \log p\left(\mathrm{y}^* = c_k \mid \mathbf{x}^*,\, \mathcal{D}\right). \tag{2.13}$$

In this thesis, we employ homoscedastic likelihood functions for regression. The mean is parametrised by a NN, and the variance is learned as a stand-alone parameter:

$$p\left(\mathbf{y}^* \mid \mathbf{x}^*,\, \boldsymbol{\xi}\right) = \mathcal{N}\big(\mathbf{y} \,\big|\, f(\mathbf{x}^*, \boldsymbol{\xi}),\, \boldsymbol{\sigma}^2 \cdot \mathbf{I}\big). \tag{2.14}$$

Thus, marginalising over $\boldsymbol{\xi}$ induces a Gaussian mixture over outputs. We approximate this mixture with a single Gaussian using moment matching, as in (Lakshminarayanan et al., 2017):

$$p\left(\mathbf{y}^* \mid \mathbf{x}^*\right) \approx \mathcal{N}\big(\mathbf{y}^* \,\big|\, \tilde{\boldsymbol{\mu}},\, \tilde{\boldsymbol{\sigma}}^2\big). \tag{2.15}$$

The mean be computed (either via MC or exactly, depending on the nature of $\boldsymbol{\xi}$) as:

$$\tilde{\boldsymbol{\mu}} = \mathbb{E}_{q(\boldsymbol{\xi})}\left[f(\mathbf{x}^*, \boldsymbol{\xi})\right] \tag{2.16}$$

$$\approx \frac{1}{M} \sum_{m=0}^{M} f(\mathbf{x}^*, \boldsymbol{\xi}_m); \quad \boldsymbol{\xi}_m \sim q(\boldsymbol{\xi}). \tag{2.17}$$

The predictive variance is obtained as the variance of the mixture distribution:

$$\tilde{\boldsymbol{\sigma}}^2 \approx \underbrace{\frac{1}{M} \sum_{m=1}^{M} f(\mathbf{x}^*, \boldsymbol{\xi}_m)^2 - \tilde{\boldsymbol{\mu}}^2}_{\text{(A)}} + \underbrace{\boldsymbol{\sigma}^2}_{\text{(B)}}; \quad \boldsymbol{\xi}_m \sim q(\boldsymbol{\xi}). \tag{2.18}$$

Here, (A) reflects model uncertainty—our lack of knowledge about $\boldsymbol{\xi}$—while (B) tells us about the irreducible uncertainty—i.e., noise—in our training data.

2.1.4 Evaluating Uncertainty Estimates

In the previous section, we discussed how to compute uncertainties for NNs. In this section, we briefly discuss how to evaluate the quality of these uncertainties. We consider the following metrics for measuring the quality of uncertainty estimates:

- **Test NLL** (lower is better): Conceptually, this metric tells us how probable it is that the (test) data was generated by our model and parameters. More precisely, it measures how well our model's predictive distribution matches the observed distribution of the test data. It is a proper scoring rule (Gneiting and Raftery, 2007), i.e., the expected NLL of a NN f_1 will be lower than a NN f_2 if and only if f_1 provides a predictive distribution that is more similar to the observed data distribution. NLL depends on both the accuracy of predictions and the calibration of their uncertainty.

- **Brier Score** (lower is better): This is another proper scoring rule that measures the accuracy of predictive probabilities in classification tasks. It is computed as the mean squared error between predicted class probabilities and one-hot class labels:

$$\mathrm{BS} = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{K}\sum_{k=1}^{K}\left(p\left(y^* = c_k \,|\, \mathbf{x}^*, \mathcal{D}\right) - \mathbb{1}[y^* = c_k]\right)^2 \tag{2.19}$$

  Erroneous predictions made with high confidence are penalised less by the Brier score than by log-likelihood. This can prevent outlier inputs from having a dominant effect on experimental results.

- **Expected Calibration Error (ECE)** (lower is better): This metric measures the difference between predictive confidence and empirical accuracy in classification. It is computed by dividing the [0,1] range into a set of bins $\{B_s\}_{s=1}^{S}$ and weighing the miscalibration in each bin by the number of points that fall into it $|B_s|$:

$$\mathrm{ECE} = \sum_{s=1}^{S}\frac{|B_s|}{N}|\mathrm{acc}(B_s) - \mathrm{conf}(B_s))|, \tag{2.20}$$

  where,

$$\mathrm{acc}(B_s) = \frac{1}{|B_s|}\sum_{\mathbf{x}\in B_s}\mathbb{1}[\mathbf{y} = \arg\max_{c_k} p\left(\mathbf{y} \,|\, \mathbf{x}, \mathcal{D}\right)], \quad \text{and} \tag{2.21}$$

$$\mathrm{conf}(B_s) = \frac{1}{|B_s|}\sum_{\mathbf{x}\in B_s}\max p\left(\mathbf{y} \,|\, \mathbf{x}, \mathcal{D}\right). \tag{2.22}$$

  ECE is not a proper scoring rule. A perfect ECE score can be obtained by predicting the marginal distribution of class labels $p(\mathbf{y})$ for every input. A well-calibrated predictor with poor accuracy would obtain low log-likelihood values but also low

ECE. Although ECE works well for binary classification, the naive adaptation to the multi-class setting suffers from several pathologies (Nixon et al., 2019).

Please see Ovadia et al. (2019); Ashukha et al. (2020) for additional discussion on evaluating uncertainty estimates of predictive models, including extensive benchmarking, potential issues with these metrics, and discussion of several additional metrics.

2.2 Deep Generative Models

Deep generative models are another approach to combining probabilistic inference with deep learning. The difference is in what is being inferred. In Section 2.1.2, inference was over the parameters $\boldsymbol{\theta}$ or the structure of a NN. In the case of deep generative models, inference is performed for the data $\mathbf{x}$ and/or a set of latent variables $\mathbf{z}$. In a deep generative model, the probability distributions for these variables are parameterised by the outputs of a NN. There are a number of successful approaches to deep generative modelling, which include the following:

1. Auto-regressive models—such as NADE, RNADE, and MADE (Larochelle and Murray, 2011; Uria et al., 2013; Germain et al., 2015), PixelCNN and PixelRNN (van den Oord et al., 2016b,c), and WaveNet (van den Oord et al., 2016a)—learn a conditional distribution $p_{\boldsymbol{\theta}}(\mathrm{x}_i \,|\, \{\mathrm{x}_j\}_{j<i})$. These models have the advantage that $p_{\boldsymbol{\theta}}(\mathbf{x})$ can be easily evaluated. However, this comes at a computational cost that scales with the dimensionality of the data.

2. Energy-based Models (EBMs)—such as restricted and deep Boltzmann machines (Smolensky, 1986; Salakhutdinov and Hinton, 2009)—learn distributions of the form

$$p(\mathbf{x}) = \frac{\exp\left(-E_{\boldsymbol{\theta}}(\mathbf{x})\right)}{Z(\boldsymbol{\theta})}, \tag{2.23}$$

where $E_{\boldsymbol{\theta}}(\mathbf{x})$ is an *energy function* which maps each observation to a scalar "energy", and $Z(\boldsymbol{\theta}) = \int \exp\left(-E_{\boldsymbol{\theta}}(\mathbf{x})\right) \mathrm{d}\mathbf{x}$ is the *partition function* or normalizing constant. EBMs are often trained via the gradient

$$\nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}) = \mathbb{E}_{p(\mathbf{x})}\left[\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x})\right] - \nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x}), \tag{2.24}$$

where the expectation can be approximated using MCMC (Hinton, 2002). In general, it is not possible to evaluate $p(\mathbf{x})$ for the EBM because calculating $Z(\boldsymbol{\theta})$

is intractable. Despite these challenges, EBMs have seen a resurgence in recent years, with Grathwohl et al. (2020) simultaneously achieving state-of-the-art results for image generation and classification at the time. Grathwohl et al. (2020) also highlight a strength of EBMs: the flexibility to choose almost any function, e.g., a NN classifier, as the energy function. EBMs are also useful for applications in which the normalising constant does not need to be evaluated, e.g., Out-of-distribution (OOD)-detection (Liu et al., 2020).

3. Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) are generative models that do not allow for $p(\boldsymbol{x})$ to be evaluated but can produce high quality samples; see (Brock et al., 2019). GANs are trained by playing a min-max game between two networks: a discriminator D and a generator G. The discriminator aims to classify whether examples are *real* or *fake* (sampled from the generator). The generator aims to produce realistic samples which *trick* the discriminator. The GAN training objective of Goodfellow et al. (2014) is

$$\min_G \max_D \mathbb{E}_{p_d(\mathbf{x})} \left[\log D(\mathbf{x})\right] + \mathbb{E}_{p_{\mathbf{z}}(\mathbf{z})} \left[\log(1 - D(G(\mathbf{z})))\right], \tag{2.25}$$

where p_d is the true data distribution, and $p_{\mathbf{z}}$ is a prior (noise) distribution. Minimizing this loss with an optimal discriminator $D(\mathbf{x}) = \frac{p_d(\mathbf{x})}{p_d(\mathbf{x})+p_g(\mathbf{x})}$ is equivalent to minimising the Jensen-Shannon Divergence

$$\mathbb{D}_{\text{JS}} \left[p_d \,||\, p_g\right] = \mathbb{D}_{\text{KL}} \left[p_d \,||\, p_m\right] + \mathbb{D}_{\text{KL}} \left[p_g \,||\, p_m\right] \tag{2.26}$$

between the true data distribution $p_d(\mathbf{x})$ and the distribution implicitly defined by the generator $p_g(\mathbf{x}) = p_g(G(\mathbf{z}))$, where $p_m(\mathbf{x}) = \frac{p_d(\mathbf{x})+p_g(\mathbf{x})}{2}$ is the mean of these two distributions. However, in practice, the discriminator is not optimal. In fact, training the discriminator to optimality can cause the gradients to the generator to vanish (Arjovsky et al., 2017). More recent GAN variants, such as the Wasserstein GAN (Arjovsky et al., 2017)—which minimises the Wasserstein distance between p_g and p_d rather than an approximation of the Jensen-Shannon Divergence—solve this problem.

4. (Denoising) diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020), also known as score-based generative models (Song et al., 2021), are a class of generative model that avoid the problem of estimating $p_{\boldsymbol{\theta}}(\mathbf{x})$ by instead estimating the *score function* $\nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\mathbf{x})$. This is done by training a NN to "denoise" samples drawn from a random walk. The random walk is defined by a sequence of *diffusion steps*

applied to samples from the training data. Concretely, we can start with a sample $\mathbf{x}_0 \sim p_d(\mathbf{x})$ and then apply T diffusion steps to obtain $\mathbf{x}_T$. The diffusion steps are defined as

$$\mathbf{x}_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t} \cdot \boldsymbol{\epsilon}_t, \quad \boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{2.27}$$

If we choose $\{\beta_t\}_{t=1}^T$ well, then $\mathbf{x}_T$ will be pure noise. I.e., we can say $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The diffusion model training objective simplifies to

$$||\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)||^2, \tag{2.28}$$

where $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ is the output of a NN (Ho et al., 2020). While this objective is conceptually and computationally simple, it is equivalent to doing VI in a latent variable model where $\{\mathbf{x}_i\}_{i<T-1}$ are the latent variables. Diffusion models are a key component in the state-of-the-art text-to-image models DALLE-2 (Ramesh et al., 2022) and Imagen (Saharia et al., 2022).

5. NFs—such as deep density models (Rippel and Adams, 2013), NICE (Dinh et al., 2015), and RealNVP (Dinh et al., 2017)—are generative models which learn a mapping from a simple base distribution (e.g., a Gaussian) to the data distribution. The mapping is parameterised by an invertible NN. This allows for $p_{\boldsymbol{\theta}}(\mathbf{x})$ to be computed exactly via a change of variables. However, the Jacobian of the NN must be computed, which can be expensive.

6. VAEs (Kingma and Welling, 2014; Rezende et al., 2014), like NFs, learn a mapping from a base distribution to the data distribution. However, unlike NFs, the mapping NN is not invertible. Instead, similar to diffusion models, amortised VI is applied to the latent variable $\mathbf{z}$. For a VAE this is achieved by learning an *inference* network (encoder) $q(\mathbf{z} \mid \mathbf{x})$, in conjunction with a *generative* network (decoder) $p(\mathbf{x} \mid \mathbf{z})$. Like a GAN, a VAE can be sampled from by first sampling $\mathbf{z}$ from the prior $p(\mathbf{z})$, often chosen to be Gaussian, and then drawing samples from the generative network $\mathbf{x} \sim p(\mathbf{x} \mid \mathbf{z})$. Unfortunately, the evidence cannot be calculated for a VAE, and we are therefore unable to perform standard maximum likelihood learning. Thus, we must resort to an alternative training objective, as discussed in Section 2.2.2.

Note that none of these methods include strong inductive biases about how the data are generated, which is arguably a key component of generative modelling (Welling, 2019). Instead, through their reliance on flexible NNs, these methods rely on inductive biases such as smoothness and compact representations.

Next, we discuss NFs and VAEs in more detail, as they will appear later in this thesis.

2.2.1 Normalising Flows

Flow-based generative models, often called *Normalising Flows*, use invertible functions (bijectors) to transform samples $\mathbf{z}$ from a base distribution $p_{\mathbf{z}}(\mathbf{z})$, often chosen to be an isotropic Gaussian, into samples $\mathbf{x}$ from a target distribution $p_{\mathbf{x}}(\mathbf{x})$

$$\mathbf{x} = f_{\boldsymbol{\theta}}(\mathbf{z}), \tag{2.29}$$

where f is usually a simple neural network parameterised by $\boldsymbol{\theta}$. Using the change of variables formula, the density $p(\mathbf{x})$ can be calculated as

$$p_{\mathbf{x}}(\mathbf{x}) = p_{\mathbf{z}}(\mathbf{z}) |\det J_f(\mathbf{z})|^{-1}, \tag{2.30}$$

where $\mathbf{z} = f^{-1}(\mathbf{x})$ and J_f is the Jacobian of f. Importantly, (2.30) can be rewritten as

$$p_{\mathbf{x}}(\mathbf{x}) = p_{\mathbf{z}}(f^{-1}(\mathbf{x})) |\det J_{f^{-1}}(\mathbf{x})|, \tag{2.31}$$

which allows us to evaluate the density of samples from $p_{\mathbf{x}}(\mathbf{x})$. A key property of bijectors is that their compositions are also bijectors. That is

$$f = f_1 \circ f_2 \circ \ldots \circ f_T \tag{2.32}$$

is a bijector if $\{f_t\}_1^T$ are bijectors. Now, the inverse and Jacobian determinant are

$$f^{-1} = f_1^{-1} \circ f_2^{-1} \circ \ldots \circ f_T^{-1}, \tag{2.33}$$

$$\det J_f(\mathbf{z}) = \prod_{t=1}^{T} \det J_{f_t}(f_{1:t-1}(\mathbf{z})), \tag{2.34}$$

where $f_{1:t-1}$ is the composition of $\{f_i\}_1^{t-1}$, and $f_{1:0}$ is the identity. As a result of these identities, it is possible to build complicated transformations as the composition of several simple transformations.

Nevertheless, a challenge when building flow-based models is that the family of functions f_t must be chosen carefully if the model is to be trained efficiently. More specifically, if f_t is an arbitrary function, then the calculation of $\det J_f(\mathbf{z})$ will in general be an $\mathcal{O}(D^3)$ operation, where D is the dimensionality of the data. By placing restrictions, such as partitioned input dimensions (Dinh et al., 2015) or using rank-1 weight matrices

(Rezende and Mohamed, 2015), on f_t, this cost can be reduced to $\mathcal{O}(D^2)$. The form of f is often still restricted despite being composed of several functions.

NFs have seen a plethora of developments and innovations. Key early works were NICE (Dinh et al., 2015) and its extension RealNVP (Dinh et al., 2017). RealNVP has been then extended with 1×1 convolutions in Glow (Kingma and Dhariwal, 2018), and autoregressive components with *inverse autoregressive flows* (Kingma et al., 2016) and *masked autoregressive flows* (Papamakarios et al., 2017) (which both increase the flexibility of the model at a computational cost that scales with the dimensionality of the data). Chen et al. (2018) show that a *neural ODE* can be used to construct *continuous normalizing flows* which can be trained efficiently and provide strong performance (Grathwohl et al., 2019). More recently, Durkan et al. (2019) proposed Neural Spline Flows (NSFs) that provide state-of-the-art results. NSFs replace the affine transformations found in RealNVP with monotonic rational quadratic spline functions, which allows for increased flexibility and better performance, sometimes matching more expensive autoregressive flows at a much lower computational cost.

NFs have also been applied to several interesting applications including VI (Rezende and Mohamed, 2015; van den Berg et al., 2018), BNNs (Louizos and Welling, 2017), semi-supervised learning (Izmailov et al., 2020; Atanov et al., 2019b), and model distillation (van den Oord et al., 2018). A thorough review of NFs can be found in Papamakarios et al. (2021).

2.2.2 Variational Autoencoders

Like NFs, VAEs sample observations $\mathbf{x}$ by first sampling latent variables

$$\mathbf{z} \sim p(\mathbf{z}) \tag{2.35}$$

and then transforming the latent variables into observations

$$\mathbf{x} \sim p(\mathbf{x} \mid \mathbf{z}). \tag{2.36}$$

Note that—unlike a NF—this is a stochastic transformation. Figure 2.1 shows the graphical model for a VAE, which consists of the generative a prior $p(\mathbf{z})$, a generative network $p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z})$ and an inference network $q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})$.

The evidence for a VAEs is intractable, thus an ELBO is used as the training objective:

$$\log p(\mathbf{x}) = \log \int p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z}) \mathrm{d}\mathbf{z} \tag{2.37}$$

$$= \log \int p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z}) \frac{q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})}{q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})} \mathrm{d}\mathbf{z} \tag{2.38}$$

$$= \log \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})} \left[p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) \frac{p(\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})} \right] \tag{2.39}$$

$$\geq \mathbb{E}_{q_{\boldsymbol{\phi}}} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) \right] - \mathbb{D}_{\mathrm{KL}} \left[q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}) \,||\, p(\mathbf{z}) \right]. \tag{2.40}$$

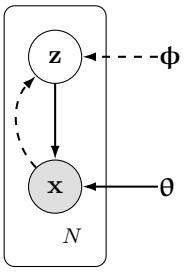


Figure 2.1: VAE generative (—) and inference (- - ·) models.

However, while this ELBO allows us to train VAEs it, by definition, implies that there is an *inference gap* between our training objective and the object of interest. We can see from the MLL:

$$\log p(\mathbf{x}) = \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})} \left[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})} \right]}_{\text{ELBO}} + \underbrace{\mathbb{D}_{\mathrm{KL}} \left[q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}) \,||\, p(\mathbf{z} \mid \mathbf{x}) \right]}_{\text{inference gap}} \tag{2.41}$$

that unless the variational distribution matches the true posterior exactly, the ELBO will not be tight. Cremer et al. (2018) show that the inference gap is not only due to the choice of variational family but also the ability of the inference network to choose good variational parameters. Additionally, they show that the inference gap can be separated into two terms: the *approximation gap* (due to the inability of $q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})$ to match $p(\mathbf{z} \mid \mathbf{x})$) and the *amortisation gap* (due to the use of amortised VI). One way to improve the performance of the generative network is the Importance Weighted Autoencoder (IWAE) (Burda et al., 2016), which uses multiple samples to better approximate the true posterior and hence reduce the size of the inference gap:

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}), \ldots, q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})} \left[\log \frac{1}{K} \sum_{k=1}^{K} \frac{p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}_k) p(\mathbf{z}_k)}{q_{\boldsymbol{\phi}}(\mathbf{z}_k \mid \mathbf{x})} \right] \geq \text{ELBO}. \tag{2.42}$$

Related to VAE, is the concept of *disentangled representations*. This refers to interpretable and factorised latent representations, such that each dimension of the latent controls a certain factor of variation in the generated data. For example, a disentangled representation of faces might have one component which controls for eye colour and another which controls for eyebrow thickness. When varied, each of these components should only change the corresponding variable in the generated face. The β-VAE (Higgins et al., 2017) modifies the objective in (2.40) by multiplying the KLD

term with a coefficient β that balances independence constraints on the elements of $\mathbf{z}$—assuming the standard isotropic Gaussian prior—with reconstruction accuracy. With $\beta > 1$, representations tend to become more disentangled. However, the notion that disentangled representations are accurate descriptions of the real-world data-generating process has been challenged by Locatello et al. (2019), who suggest that without inductive biases, unsupervised learning of disentangled representations is impossible.

VAEs are an active area of research. In fact, Kingma et al. (2016) originally proposed the inverse autoregressive flow as a way to improve the performance of VAEs by increasing the flexibility of the approximate posterior. More recently, Vahdat and Kautz (2020) proposed a hierarchical VAE which provided state-of-the-art results for image generation with VAEs (and NFs).

2.3 Summary

In this chapter, we have discussed the literature related to probabilistic approaches to deep learning. We focused on two main areas: Bayesian deep learning and deep generative models. In the case of Bayesian deep learning, we discussed the challenges of performing probabilistic inference in NNs. We highlighted several challenges, including the difficulty of choosing a good prior and the challenge of performing inference in high-dimensional spaces. Chapters 3, 4 and 5 will all focus on improving uncertainty estimation in NNs by addressing (or avoiding) these challenges. We also discussed how to compute and evaluate uncertainties in NNs. In the case of deep generative models, we discussed several different approaches to deep generative modelling, including auto-regressive models, EBMs, GANs, diffusion models, NFs, and VAEs. We noted that none of these models include strong inductive biases about how the data are actually generated, instead, they rely on flexible NNs to learn the data distribution in various ways. Chapter 6 will focus on how to incorporate stronger inductive biases into deep generative models.

In the next chapter, we will begin to address some of the challenges of uncertainty estimation in NNs by *avoiding* inference in the high-dimensional weight space of NNs altogether and instead performing inference over the depth of the network.

Chapter 3

Depth Uncertainty in Neural Networks

In this chapter, we introduce a method for inferring the depth of a NN and improving the calibration of the model's predictive uncertainty by marginalising the uncertainty over the depth. Our contributions are:

1. In Section 3.2, we introduce Depth Uncertainty Networks (DUNs) our proposed model that treats depth as a random variable, and we discuss how best to do inference in DUNs, comparing Variational Inference (VI) and exact Marginal Log Likelihood (MLL) maximisation.

2. We provide experimental validation for DUNs in Section 3.3. In particular, we validate our inference strategy in Section 3.3.1. We demonstrate that DUNs provide calibrated predictive uncertainty estimates in Sections 3.3.2 to 3.3.4. Finally, in Sections 3.3.5 and 3.3.6 we apply DUNs to Neural Architecture Search (NAS) and active learning.

This chapter is primarily based on the paper "Depth Uncertainty in Neural Networks" (Antorán et al., 2020)—written in collaboration with Javier Antorán and José Miguel Hernández-Lobato—with additional content from "Depth Uncertainty Networks for Active Learning" (Murray et al., 2021b)—with Chelsea Murray, Javier and Miguel. The original idea for Antorán et al. (2020) is due to Javier. I was heavily involved with all aspects of the project, sharing responsibility with Javier for conceptualisation, exploration, coding, evaluation, and presentation of the results, as well as writing the paper. For Murray et al. (2021b), I took a more supervisory role focusing on conceptualisation and writing, with Chelsea being responsible for all of the experimentation. Miguel provided guidance and high-level input throughout both projects.

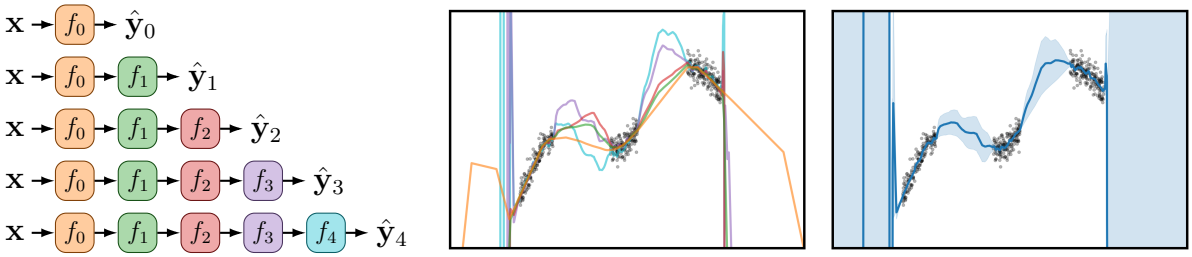


Figure 3.1: A DUN is composed of subnetworks of increasing depth (*left*, colours denote layers with shared parameters). These correspond to increasingly complex functions (*centre*, colours denote depth at which predictions are made). Marginalising over depth yields model uncertainty through disagreement of these functions (*right*, error bars denote 1 std. dev.).

3.1 Motivation

Despite the widespread adoption of deep learning, building models that provide robust uncertainty estimates remains a challenge. This is especially important for real-world applications, where we cannot expect the distribution of observations to be the same as that of the training data. Deep models tend to be pathologically overconfident, even when their predictions are incorrect (Nguyen et al., 2015; Amodei et al., 2016). If artificial intelligence systems would reliably identify cases in which they expect to underperform and request human intervention, they could more safely be deployed in medical scenarios (Filos et al., 2019) or self-driving vehicles (Levinson et al., 2011), for example.

In response, a rapidly growing subfield has emerged seeking to build uncertainty-aware NNs (Hernández-Lobato and Adams, 2015; Gal and Ghahramani, 2016; Lakshminarayanan et al., 2017). Regrettably, these methods rarely make the leap from research to production due to a series of shortcomings:

1. *Implementation Complexity:* they can be technically complicated and sensitive to hyperparameter choice.

2. *Computational cost:* they can take orders of magnitude longer to converge than regular networks or require training multiple networks. At test time, averaging the predictions from multiple models is often required.

3. *Weak performance:* they rely on crude approximations to achieve scalability, resulting in unreliable uncertainty estimates (Foong et al., 2020; Burt et al., 2021).

In this chapter, we introduce DUNs, a probabilistic model that treats the depth of a NN as a random variable over which to perform inference. In contrast to more typical weight-space approaches for Bayesian inference in NNs—e.g., as described in

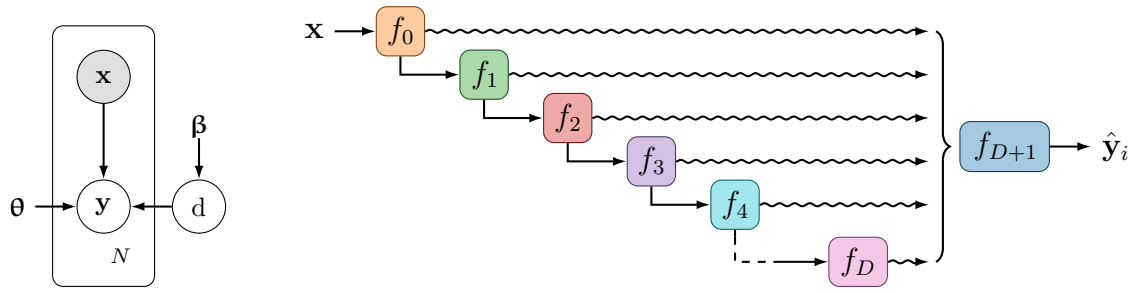


Figure 3.2: Left: graphical model under consideration. Right: computational model. Each layer's activations are passed through the output block, producing per-depth predictions.

Section 2.1.2—ours reflects a lack of knowledge about how deep our NN should be. We treat NN weights as learnable hyperparameters. In DUNs, marginalising over depth is equivalent to performing Bayesian Model Averaging (BMA) over an ensemble of progressively deeper NNs. As shown in Figure 3.1, DUNs exploit the overparametrisation of a single deep NN to generate diverse explanations of the data. The key advantages of DUNs are:

1. *Implementation simplicity*: requiring only minor additions to vanilla deep learning code, and no changes to the hyperparameters or training regime.

2. *Cheap deployment*: computing exact predictive posteriors with a single forward pass.

3. *Calibrated uncertainty*: our experiments show that DUNs are competitive with strong baselines in terms of predictive performance, Out-of-distribution (OOD) detection, and robustness to corruptions.

3.2 Depth Uncertainty Networks

Consider a dataset $\mathcal{D} = \{\mathbf{x}^{(n)}, \mathbf{y}^{(n)}\}_{n=1}^{N}$ and a NN composed of an input block $f_0(\cdot)$, D intermediate blocks $\{f_d(\cdot)\}_{d=1}^{D}$, and an output block $f_{D+1}(\cdot)$. Each block is a group of one or more stacked linear and non-linear operations. The activations at depth $d \in [0, D]$, $\mathbf{a}_d$, are obtained recursively as $\mathbf{a}_d = f_d(\mathbf{a}_{d-1})$, $\mathbf{a}_0 = f_0(\mathbf{x})$.

A forward pass through the NN is an iterative process, where each successive block $f_d(\cdot)$ refines the previous block's activation. Predictions can be made at each step of this procedure by applying the output block to each intermediate block's activations: $\hat{\mathbf{y}}_d = f_{D+1}(\mathbf{a}_d)$. This computational model is displayed in Figure 3.2. Recall, from Figure 3.1, that we can leverage the disagreement among intermediate blocks' predictions to quantify model uncertainty.

3.2.1 Probabilistic Model: Depth as a Random Variable

We place a categorical prior over NN depth $p_{\boldsymbol{\beta}}(\mathrm{d}) = \mathrm{Cat}(\mathrm{d} \mid \boldsymbol{\beta})$. Referring to NN weights as $\boldsymbol{\theta}$, we parametrise the likelihood for each depth using the corresponding subnetwork's output: $p(\mathbf{y} \mid \mathbf{x}, \mathrm{d} = d; \boldsymbol{\theta}) = p(\mathbf{y} \mid f_{D+1}(\mathbf{a}_d; \boldsymbol{\theta}))$. A graphical model is shown in Figure 3.2. For a given weight configuration, the likelihood for every depth, and thus our model's MLL,

$$\log p(\mathcal{D}; \boldsymbol{\theta}) = \log \sum_{d=0}^{D} \left(p_{\boldsymbol{\beta}}(\mathrm{d} = d) \cdot \prod_{n=1}^{N} p\left(\mathbf{y}^{(n)} \mid \mathbf{x}^{(n)}, \mathrm{d} = d; \boldsymbol{\theta}\right) \right), \tag{3.1}$$

can be obtained with a *single forward pass* over the training set by exploiting the sequential nature of feed-forward NNs. The posterior over depth,

$$p(\mathrm{d} \mid \mathcal{D}; \boldsymbol{\theta}) = \frac{p(\mathcal{D} \mid \mathrm{d}; \boldsymbol{\theta}) p_{\boldsymbol{\beta}}(\mathrm{d})}{p(\mathcal{D}; \boldsymbol{\theta})}, \tag{3.2}$$

is a categorical distribution that tells us about how well each subnetwork explains the data.

A key advantage of deep NN lies in their capacity for automatic feature extraction and representation learning. For instance, Zeiler and Fergus (2014) demonstrate that CNNs detect successively more abstract features in deeper layers. Similarly, Frosst et al. (2019) find that maximising the entanglement of different class representations in intermediate layers yields better generalisation. Given these results, using all of our network's intermediate blocks for prediction might be suboptimal. Instead, we infer whether each block should be used to learn representations or perform predictions, which we can leverage for ensembling, by treating network depth as a random variable. As shown in Figure 3.3, subnetworks too shallow to explain the data are assigned low posterior probability; they perform feature extraction.

3.2.2 Inference in DUNs

We consider learning NN weights by directly maximising (3.1) with respect to $\boldsymbol{\theta}$, using backpropagation and the *log-sum-exp* trick. However, the gradients of (3.1) reaching each subnetwork are weighted by the corresponding depth's posterior mass:

$$\frac{\partial}{\partial \boldsymbol{\theta}} \log p(\mathcal{D}; \boldsymbol{\theta}) = \frac{\partial}{\partial \boldsymbol{\theta}} \texttt{logsumexp}_d(\log p(\mathcal{D} \mid d; \boldsymbol{\theta}) + \log p(d))$$

$$
\begin{aligned}
&= \sum_{d=0}^{D} \frac{p(\mathcal{D} \mid \mathrm{d}=d;\, \boldsymbol{\theta}) p(\mathrm{d}=d)}{\sum_{d'=0}^{D} p(\mathcal{D} \mid \mathrm{d}=d';\, \boldsymbol{\theta}) p(\mathrm{d}=d')} \frac{\partial}{\partial \boldsymbol{\theta}} \log p(\mathcal{D} \mid \mathrm{d}=d;\, \boldsymbol{\theta}) \\
&= \sum_{d=0}^{D} p(\mathrm{d}=d \mid \mathcal{D};\, \boldsymbol{\theta}) \frac{\partial}{\partial \boldsymbol{\theta}} \log p(\mathcal{D} \mid \mathrm{d}=d;\, \boldsymbol{\theta}) \\
&= \mathbb{E}_{p(\mathrm{d} \mid \mathcal{D};\, \boldsymbol{\theta})} \left[\frac{\partial}{\partial \boldsymbol{\theta}} \log p(\mathcal{D} \mid \mathrm{d};\, \boldsymbol{\theta}) \right]. \qquad (3.3)
\end{aligned}
$$

This leads to local optima where all but one subnetworks' gradients vanish since the weights of the subnetwork that best explains the data at initialisation will receive larger gradients. This will result in this depth fitting the data even better and receiving larger gradients in successive iterations while the gradients for subnetworks of different depths vanish, creating a rich get richer scenario. Thus, the MLL objective is prone to hard-to-escape local optima, at which a single depth is used, or in other words, at which the posterior collapses to a delta function over an arbitrary depth, leaving us with a deterministic NN. This can be especially problematic if the initial posterior has its maximum over shallow depths, as this will reduce the capacity of the NN.

When working with large datasets, one might indeed expect the true posterior over depth to be a delta. However, because modern NNs are underspecified even for large datasets, multiple depths should be able to explain the data simultaneously (as demonstrated in Figure 3.3).

We can avoid the above pathology by decoupling the optimisation of NN weights $\boldsymbol{\theta}$ from the posterior distribution. In latent variable models, the expectation maximisation algorithm (Bishop, 2006) allows us to optimise the MLL by iteratively computing $p(\mathrm{d} \mid \mathcal{D};\, \boldsymbol{\theta})$ and then updating $\boldsymbol{\theta}$. We propose to use stochastic gradient VI as an alternative more amenable to NN optimisation. We introduce a surrogate categorical distribution over depth $q_{\boldsymbol{\alpha}}(\mathrm{d}) = \mathrm{Cat}(\mathrm{d} \mid \boldsymbol{\alpha})$, and use this to construct a lower bound on (3.1):

$$
\begin{aligned}
\log p(\mathcal{D};\, \boldsymbol{\theta}) &= \log \sum_{d} \left[p(\mathcal{D},\, \mathrm{d}=d;\, \boldsymbol{\theta}) \right] \\
&= \log \left[\sum_{d} p(\mathcal{D},\, d;\, \boldsymbol{\theta}) \frac{q_{\boldsymbol{\alpha}}(d)}{q_{\boldsymbol{\alpha}}(d)} \right] \quad \triangleright \left(\times 1 = \frac{q_{\boldsymbol{\alpha}}(d)}{q_{\boldsymbol{\alpha}}(d)} \right) \\
&= \log \left[\mathbb{E}_{q_{\boldsymbol{\alpha}}(\mathrm{d})} \frac{p(\mathcal{D} \mid \mathrm{d};\, \boldsymbol{\theta}) p_{\boldsymbol{\beta}}(\mathrm{d})}{q_{\boldsymbol{\alpha}}(\mathrm{d})} \right] \\
&\geq \mathbb{E}_{q_{\boldsymbol{\alpha}}(\mathrm{d})} \left[\log \frac{p(\mathcal{D} \mid \mathrm{d};\, \boldsymbol{\theta}) p_{\boldsymbol{\beta}}(\mathrm{d})}{q_{\boldsymbol{\alpha}}(\mathrm{d})} \right] \quad \triangleright \text{(Jensen's Inequality)}
\end{aligned}
$$

$$= \sum_{n=1}^{N} \mathbb{E}_{q_{\boldsymbol{\alpha}}(\mathrm{d})} \left[\log p\left(\mathbf{y}^{(n)} \,\middle|\, \mathbf{x}^{(n)}, \mathrm{d}; \boldsymbol{\theta}\right) \right] - \mathbb{D}_{\mathrm{KL}} \left[q_{\boldsymbol{\alpha}}(\mathrm{d}) \,||\, p_{\boldsymbol{\beta}}(\mathrm{d}) \right]$$
$$= \mathcal{L}(\boldsymbol{\alpha}, \boldsymbol{\theta}). \tag{3.4}$$

This Evidence Lower BOund (ELBO) allows us to optimise the variational parameters $\boldsymbol{\alpha}$ and network weights $\boldsymbol{\theta}$ simultaneously using gradients. Because both our variational and true posteriors are categorical, (3.4) is convex with respect to $\boldsymbol{\alpha}$. At the optima, $q_{\boldsymbol{\alpha}}(\mathrm{d}) = p(\mathrm{d} \,|\, \mathcal{D}; \boldsymbol{\theta})$ and the bound is tight. Thus, we perform exact rather than approximate inference.

In contrast to optimisation of (3.1), VI decouples the likelihood at each depth from the approximate posterior during optimisation:

$$\frac{\partial}{\partial \boldsymbol{\theta}} \mathcal{L} = \sum_{d=0}^{D} q_{\boldsymbol{\alpha}}(\mathrm{d} = d) \frac{\partial}{\partial \boldsymbol{\theta}} \log p\left(\mathcal{D} \,|\, \mathrm{d} = d; \boldsymbol{\theta}\right) \tag{3.5}$$

$$\frac{\partial}{\partial \boldsymbol{\alpha}_d} \mathcal{L} = \log p\left(\mathcal{D} \,|\, \mathrm{d} = d; \boldsymbol{\theta}\right) \frac{\partial}{\partial \boldsymbol{\alpha}_d} q_{\boldsymbol{\alpha}}(\mathrm{d} = d) - \left(\log q_{\boldsymbol{\alpha}}(d) - \log p_{\boldsymbol{\beta}}(d) + 1\right) \frac{\partial}{\partial \alpha_i} q_{\boldsymbol{\alpha}}(d) \tag{3.6}$$

For moderate to large datasets, when updating the variational parameters $\boldsymbol{\alpha}$, the data dependent term of the ELBO's gradient will dominate. However, the gradients that reach the variational parameters are scaled by the log-likelihood at each depth. In contrast, in (3.3), the posterior for each depth scales the gradients directly. We conjecture that, with VI, $\boldsymbol{\alpha}$ will converge more slowly than the true posterior when optimising the MLL directly. This allows network weights to reach solutions that explain the data well at multiple depths. In Section 3.3.1, we show that this does seem to be the case.

In addition to making optimisation easier, $\mathbb{E}_{q_{\boldsymbol{\alpha}}(\mathrm{d})} \left[\log p\left(\mathbf{y} \,|\, \mathbf{x}, \mathrm{d}; \boldsymbol{\theta}\right) \right]$ can be computed from the activations at every depth. Consequently, both terms in (3.4) can be evaluated exactly, with only a single forward pass. This removes the need for high variance Monte Carlo (MC) gradient estimators, often required by VI methods for NNs. When using mini-batches of size B, we stochastically estimate the ELBO in (3.4) as

$$\mathcal{L}(\boldsymbol{\alpha}, \boldsymbol{\theta}) \approx \frac{N}{B} \sum_{n=1}^{B} \sum_{d=0}^{D} \left(\log p\left(\mathbf{y}^{(n)} \,\middle|\, \mathbf{x}^{(n)}, \mathrm{d} = d; \boldsymbol{\theta}\right) \cdot \boldsymbol{\alpha}_d \right) - \sum_{d=0}^{D} \left(\boldsymbol{\alpha}_d \log \frac{\boldsymbol{\alpha}_d}{\boldsymbol{\beta}_d} \right). \tag{3.7}$$

Predictions for new data $\mathbf{x}^*$ are made by marginalising depth with the variational posterior:

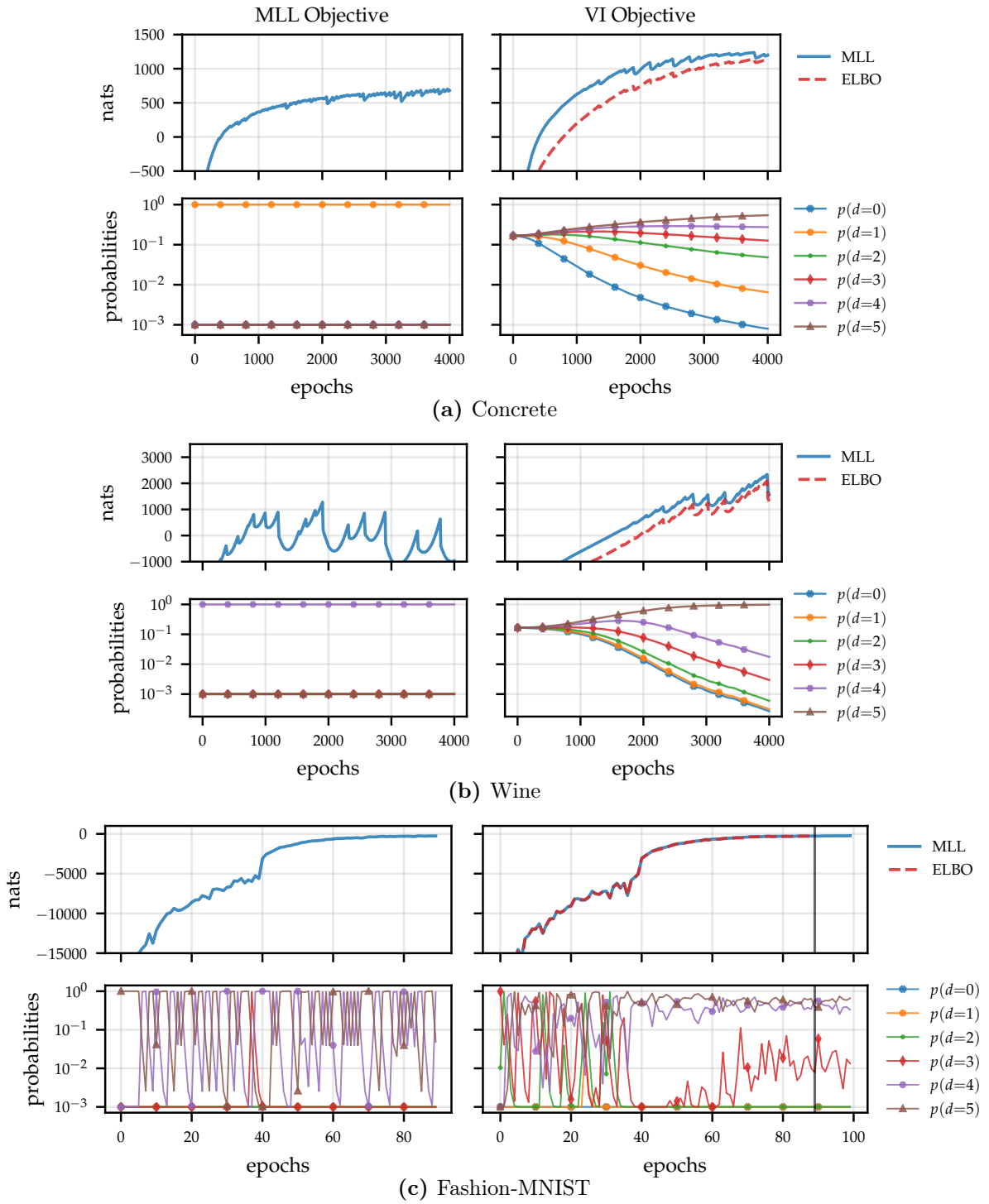
$$p(\mathbf{y}^* \,|\, \mathbf{x}^*, \mathcal{D}; \boldsymbol{\theta}) = \sum_{d=0}^{D} p(\mathbf{y}^* \,|\, \mathbf{x}^*, \mathrm{d} = d; \boldsymbol{\theta}) q_{\boldsymbol{\alpha}}(\mathrm{d} = d). \tag{3.8}$$

3.3 Experiments

First, we compare the MLL and VI training approaches for DUNs. We then evaluate DUNs on toy-regression, real-world regression, and image classification tasks. As baselines, we provide results for vanilla NNs (denoted as 'Stochastic Gradient Descent (SGD)'), MC Dropout (Gal and Ghahramani, 2016), and deep ensembles (Lakshminarayanan et al., 2017), arguably the strongest approach for uncertainty estimation in deep learning (Ovadia et al., 2019; Ashukha et al., 2020). For regression tasks, we also include Gaussian Mean-Field Variational Inference (MFVI) (Blundell et al., 2015) with the local reparametrisation trick (Kingma et al., 2015). For image robustness tasks, we also compare against Stochastic-ResNets (Huang et al., 2016) and deep ensembles of networks with multiple depths. We study all methods in terms of accuracy, uncertainty quantification, and robustness to corrupted or OOD data. We place a uniform prior over DUN depth. See Section 2.1.3, Section 2.1.4, and Section A.2 for detailed descriptions of the techniques we use to compute and evaluate uncertainty estimates, and our experimental setup, respectively. Code is available at `https://github.com/cambridge-mlg/DUN`.

3.3.1 Comparing MLL and VI training

Figures 3.3a and 3.3b compare the optimisation of a 5 hidden layer fully connected DUN on the concrete, and wine datasets using estimates of the MLL (3.1) and ELBO (3.7). In both cases, the former approach converges to a local optima where all but one depth's probabilities go to 0. Note that while the posterior, in Figure 3.3b, collapses to a non-zero depth, we found that a depth of 0 was most common in collapsed posteriors. With VI, the surrogate posterior converges more slowly than the network weights. This allows $\boldsymbol{\theta}$ to reach a configuration where multiple depths can be used for prediction. Towards the end of training, the variational gap vanishes. The surrogate distribution approaches the true posterior without collapsing to a delta. The MLL values obtained with VI are larger than those obtained with (3.1), i.e., our proposed approach finds better explanations for the data.



(a) Concrete

(b) Wine

(c) Fashion-MNIST

Figure 3.3: Top row: progression of MLL and ELBO during training. Bottom: progression of all six depth posterior probabilities. The left column corresponds to optimising the MLL directly and the right to VI. For the latter, variational posterior probabilities $q(\mathrm{d})$ are shown, and the MLL is calculated but not used for optimisation.

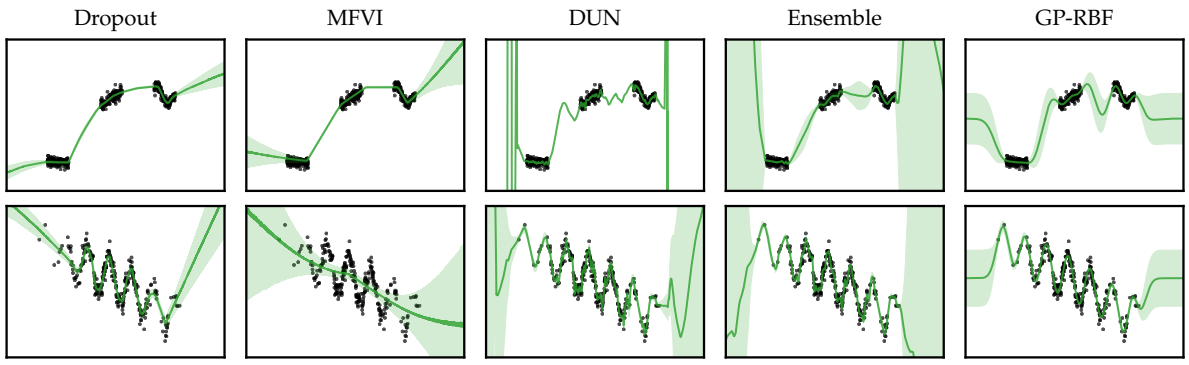


Figure 3.4: Top row: toy dataset from Izmailov et al. (2019). Bottom: Wiggle dataset. Black dots denote data points. Error bars represent the standard deviation among mean predictions.

In Figure 3.3c, both training schemes obtain very similar MLL values. The dataset under consideration is much larger than the one in Figures 3.3a and 3.3b, but the dimensionality of the latent variable stays the same. Hence, the variational gap is small relative to the MLL. Nevertheless, unlike with the MLL objective, VI training results in posteriors that avoid placing all of their mass on a single depth setting. Additionally, in Figure 3.3c, we optimise (3.1) after reaching a local optima with (3.7). This does not cause posterior collapse, showing that MLL optimisation's poor performance is due to a propensity for poor local optima.

3.3.2 Toy Datasets

In Figure 3.4, we consider two synthetic 1D datasets and compare DUNs with baselines. We use 3-hidden-layer, 100-hidden-unit, fully connected networks with residual connections for the baselines. DUNs use the same architecture but with 15 hidden layers. Gaussian Processes (GPs) use the RBF kernel. We found these configurations to work well empirically.

The first dataset, which is taken from Izmailov et al. (2019), contains three disjoint clusters of data. Both MFVI and Dropout present error bars that are similar in the data-dense and in-between regions. MFVI underfits slightly, not capturing smoothness in the data. DUNs perform most similarly to Ensembles. They are both able to fit the data well and express in-between uncertainty. Their error bars become large very quickly in the extrapolation regime.

Our second dataset consists of 300 samples from $y = \sin(\pi x) + 0.2\cos(4\pi x) - 0.3x + \epsilon$, where $\epsilon \sim \mathcal{N}(0, 0.25)$ and $x \sim \mathcal{N}(5, 2.5)$. We dub it "Wiggle". Dropout struggles to fit this faster varying function outside of the data-dense regions. MFVI fails completely.

DUNs and Ensembles both fit the data well and provide error bars that grow as the data becomes sparse.

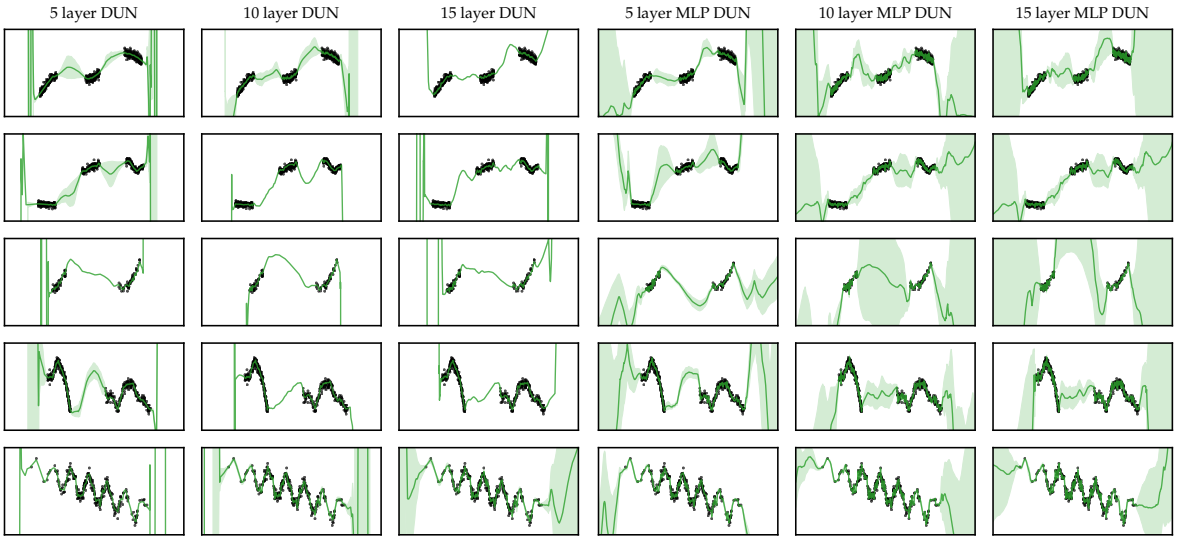


Figure 3.5: DUN depth experiments. Left 3 columns: DUNs with residual connections. Right 3 columns: DUNs without residual connections. Row 1: a simple dataset made up of 3 disjoint clusters (Simple_1d). Row 2: taken from (Izmailov et al., 2019). Row 3: taken from (Foong et al., 2019). Row 4: generated by sampling from a GP with a Matern kernel. Row 5: Wiggle dataset.

In the above experiments, we used DUNs with more layers than the baselines because DUN performance benefits from depth. This is due to two reasons. Firstly, increased depth means increasing the number of explanations of the data which are marginalised. Secondly, deeper subnetworks are able to express faster varying functions, which are more likely to disagree with each other. Figure 3.5 shows that performance increases with depth. However, often 5 layers are sufficient to produce reasonable, while slightly smaller, uncertainty estimates. Figure 3.5 also shows results for DUNs without residual connections. Here we see that the uncertainty estimates, especially for shallower models, are much smaller—particularly in the extrapolation regime.

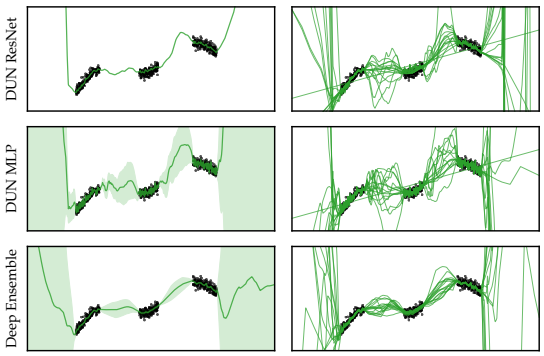


Figure 3.6: Comparison of DUN and ensemble fits for the Simple_1d dataset with 15 layer DUNs, and 20× 3-hidden-layer ensemble elements. Left: mean predictions and standard deviations. Right: individual predictions. For DUNs, the individual predictions are weighted by $q_{\boldsymbol{\alpha}}(d)$, whereas ensembles use an equal weighting.

We further compare the in-distribution fits from residual DUNs, Multi-Layer Perceptron (MLP) DUNs, and deep ensembles in Figure 3.6. Ensemble elements differ slightly from each other in their predictions within the data-dense regions. These predictions are averaged, making for mostly smooth functions. Functions expressed at most depths of the MLP DUNs seem to vary together rapidly within the data region. Their mean prediction also varies rapidly, suggesting overfitting. In an MLP architecture, each successive layer only receives the previous one's output as its input. We hypothesise that, because of this structure, once a layer overfits to a data point, the following layer is unlikely to modify the function in the area of that data point, as that would increase the training loss. This leads to most subnetworks only disagreeing about their predictions out of distribution. Functions expressed by residual DUNs differ somewhat in distribution, allowing some robustness to overfitting. We hypothesise that this occurs because each layer takes a linear combination of all previous layers' activations as its input. This prevents reusing the previous subnetworks' fits. Ensembles provide diverse explanations both in and out of distribution. This results in both better accuracy and predictive uncertainty than single models. DUNs provide explanations which differ from each other mostly OOD. They provide uncertainty estimates OOD, but their accuracy on in-distribution points is similar to that of a single model.

3.3.3 Tabular Regression

We evaluate DUNs alongside NN-based baselines on UCI regression datasets using standard and gap splits as in Hernández-Lobato and Adams (2015); Foong et al. (2019). Although we follow standard procedure to compare only with NN baselines (Lakshminarayanan et al., 2017; Foong et al., 2019), we believe that GPs would make an interesting point of comparison due to their exact inference in these small-scale settings. We encourage future researchers to include them as a baseline. We also use the large-scale non-stationary flight delay dataset, preprocessed by Hensman et al. (2013). Following Deisenroth and Ng (2015), we train on the first 2M data points and test on the subsequent 100k. We select all hyperparameters, including NN depth, using Bayesian optimisation with HyperBand (Falkner et al., 2018). See Section A.2.2 for details. We evaluate methods with Root Mean Squared Error (RMSE), Log Likelihood (LL), Regression Calibration Error (RCE), and Tail Calibration Error (TCE). The latter two measures are novel:

- **Regression Calibration Error (RCE)** (lower is better): We extend the idea of Expected Calibration Error (ECE) to regression settings. We seek to assess how

well our model's predictive distribution describes the residuals obtained on the test set. It is not straightforward to define bins, like in standard ECE, because our predictive distribution might not have finite support. We apply the CDF of our predictive distribution to our test targets. If the predictive distribution describes the targets well, the transformed distribution should resemble a uniform with support $[0, 1]$. This procedure is common for back testing market risk models (Dowd, 2013).

To assess the global similarity between our targets' distribution and our predictive distribution, we separate the $[0, 1]$ interval into S equal-sized bins $\{B_s\}_{s=1}^{S}$. We compute calibration error in each bin as the difference between the proportion of points that have fallen within that bin and $1/S$:

$$\text{RCE} = \sum_{s=1}^{S} \frac{|B_s|}{N} \cdot \left| \frac{1}{S} - \frac{|B_s|}{N} \right|; \quad |B_s| = \sum_{n=1}^{N} \mathbb{1}[CDF_{p(\mathrm{y}\,|\,\mathbf{x}^{(n)})}(\mathrm{y}^{(n)}) \in B_s] \tag{3.9}$$

- **Tail Calibration Error (TCE)** (lower is better): Alternatively, we can assess how well our model predicts extreme values with a "frequency of tail losses" approach (Kupiec, 1995). It might not be realistic to assume the noise in our targets is Gaussian. Only considering calibration at the tails of the predictive distribution allows us to ignore the shape mismatch between the predictive distribution and the true distribution over targets. Instead, we focus on our model's capacity to predict which inputs it is likely to make large mistakes. This can be used to ensure our model is not overconfident OOD. We specify two bins $\{B_0, B_1\}$, one at each tail end of our predictive distribution, and compute TCE as:

$$\text{TCE} = \sum_{s=0}^{1} \frac{|B_s|}{|B_0| + |B_1|} \cdot \left| \frac{1}{\tau} - \frac{|B_s|}{N} \right|;$$

$$|B_0| = \sum_{n=1}^{N} \mathbb{1}[CDF_{p(y|\mathbf{x}^{(n)})}(y^{(n)}) < \tau]; \quad |B_1| = \sum_{n=1}^{N} \mathbb{1}[CDF_{p(\mathrm{y}|\mathbf{x}^{(n)})}(\mathrm{y}^{(n)}) \geq (1-\tau)]$$

We specify the tail range of our distribution by selecting τ. Note that this is slightly different from Kupiec (1995), who uses a binomial test to assess whether a model's predictive distribution agrees with the distribution over targets in the tails.

RCE and TCE are not proper scoring rules. Additionally, they are only applicable to 1-dimensional continuous target variables.

UCI standard split results are found in Figure 3.7. For each dataset and metric, we rank methods from 1 to 5 based on mean performance. We report mean ranks and standard deviations. Dropout obtains the best mean rank in terms of RMSE, followed closely by Ensembles. DUNs are third, significantly ahead of MFVI and SGD. Even so, DUNs outperform Dropout and Ensembles in terms of TCE, i.e., DUNs more reliably assign large error bars to points on which they make incorrect predictions. Consequently, in terms of LL, a metric which considers both uncertainty and accuracy, DUNs perform competitively (the LL rank distributions for all three methods overlap almost completely). MFVI provides the best calibrated uncertainty estimates. Despite this, its mean predictions are inaccurate, as evidenced by it being last in terms of RMSE. This leads to MFVI's LL rank only being better than SGD's. Results for gap splits, designed to evaluate methods' capacity to express in-between uncertainty, Figure 3.8. Here, DUNs outperform Dropout in terms of LL rank. However, they are both outperformed by MFVI and ensembles.

The flights dataset is known for strong covariate shift between its train and test sets, which are sampled from contiguous time periods. LL values are strongly dependent on calibrated uncertainty. As shown in Table 3.1, DUNs' RMSE is similar to that of SGD, with Dropout and Ensembles performing best. Again, DUNs present superior uncertainty calibration. This allows them to achieve the best LL, tied with Ensembles and Dropout. We speculate that DUNs' calibration stems from being able to perform exact inference, albeit in depth space.

In terms of prediction time, DUNs clearly outrank Dropout, Ensembles, and MFVI on UCI. Due to depth, or maximum depth D for DUNs, being chosen with Bayesian optimisation, methods' batch times vary across datasets. DUNs are often deeper because the quality of their uncertainty estimates improves with additional explanations of the data. As a result, SGD clearly outranks DUNs. On flights, increased depth causes DUNs' prediction time to lie in between Dropout's and Ensembles'.

Table 3.1: Results obtained on the flights dataset (2M). Mean and std. dev. values are computed across 5 independent training runs.

	LL ↑	RMSE ↓	TCE ↓	Time ↓
DUN	**-4.95** ±0.01	34.69 ±0.28	0.087 ±0.009	0.026 ±0.001
Dropout	**-4.95** ±0.02	**34.28** ±0.11	0.096 ±0.017	0.016 ±0.001
Ensemble	**-4.95** ±0.01	34.32 ±0.13	0.090 ±0.008	0.031 ±0.001
MFVI	-5.02 ±0.05	36.72 ±1.84	**0.068** ±0.014	0.547 ±0.003
SGD	-4.97 ±0.01	34.61 ±0.19	0.084 ±0.010	**0.002** ±0.000

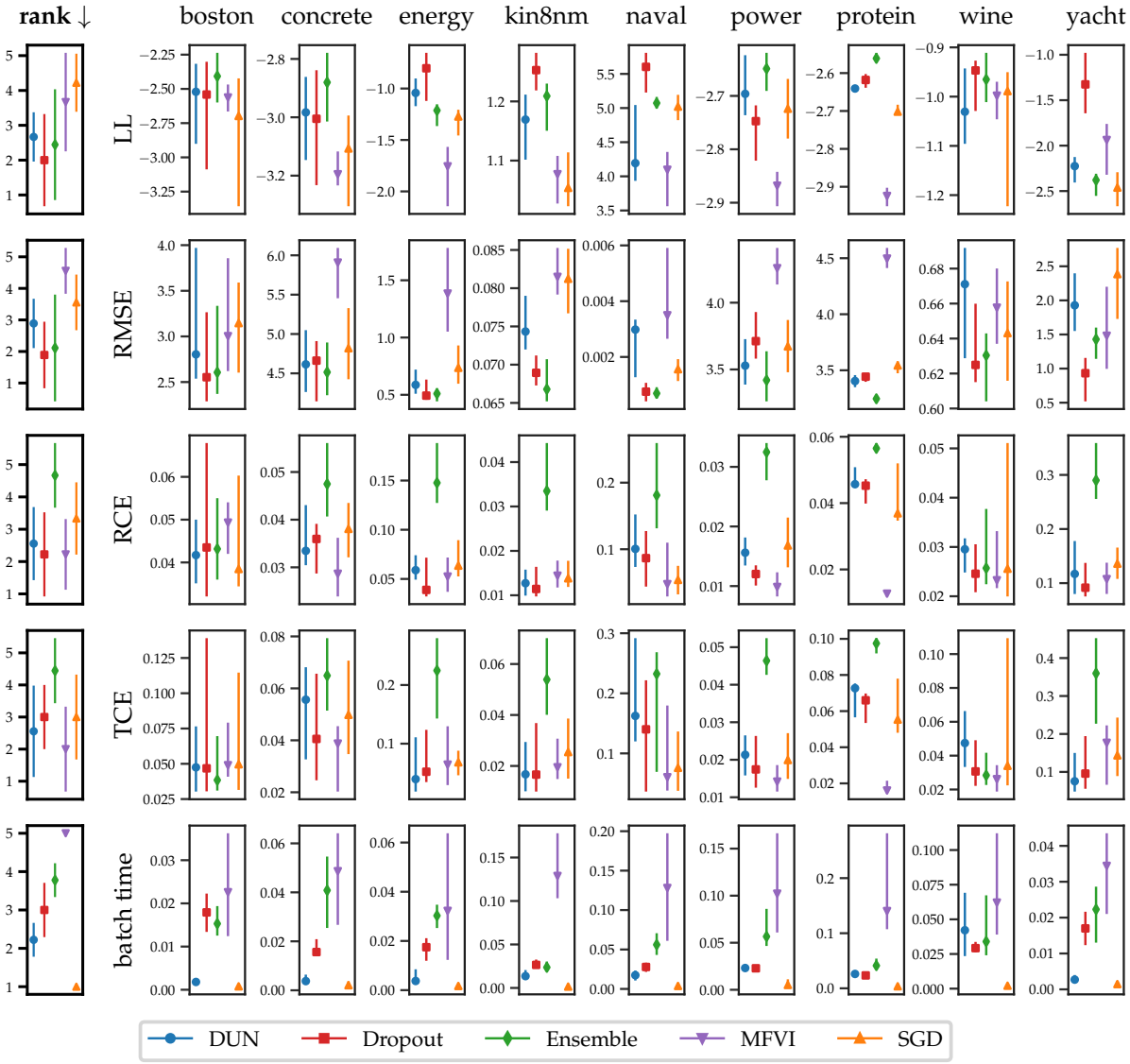


Figure 3.7: Quartiles for results on UCI regression datasets across standard splits. Average ranks are computed across datasets. For LL, higher is better. Otherwise, lower is better.

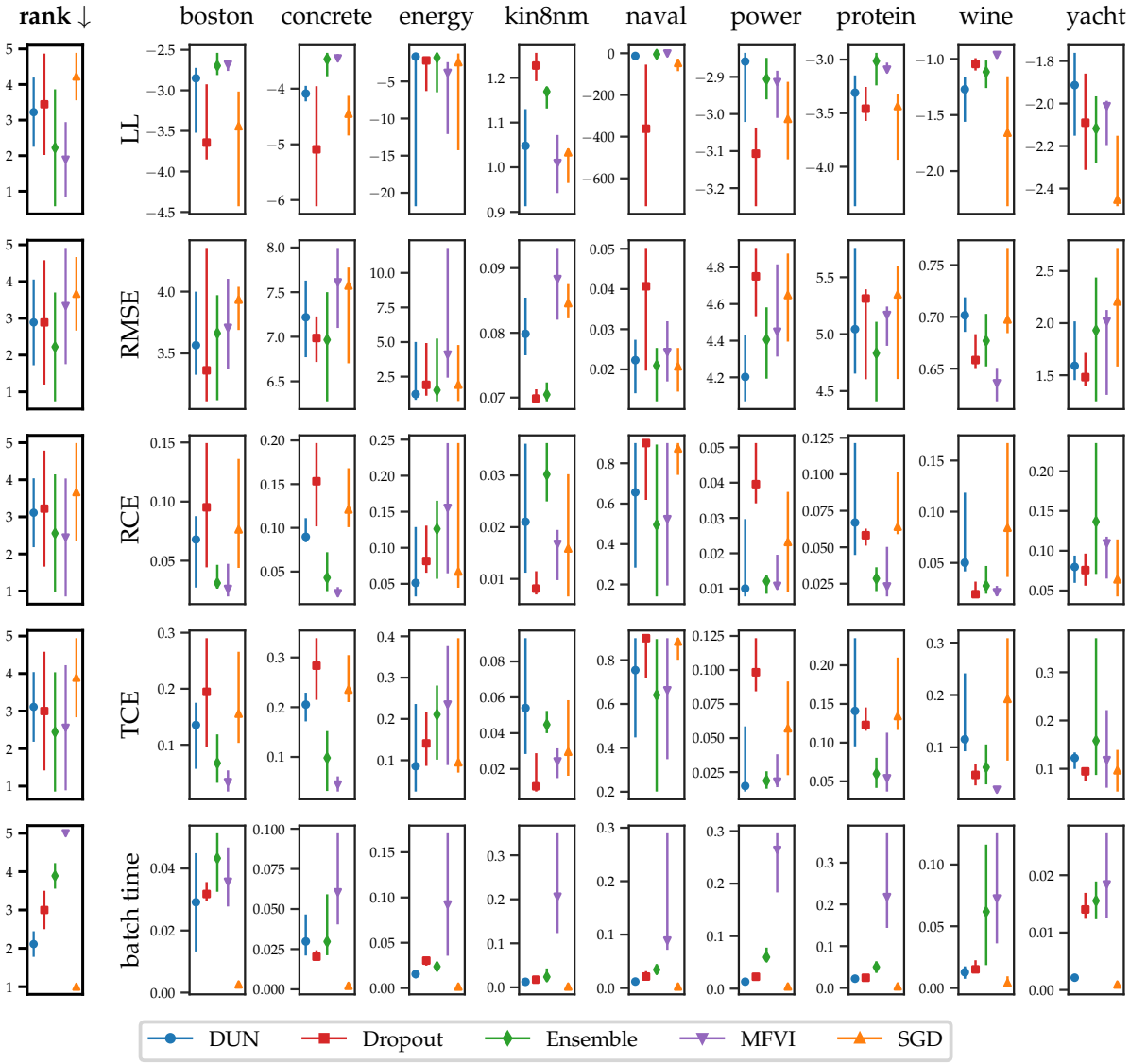


Figure 3.8: Quartiles for results on UCI regression datasets across gap splits. Average ranks are computed across datasets. For LL, higher is better. Otherwise, lower is better.

3.3.4 Image Classification

We train ResNet-50 (He et al., 2016a) using all methods under consideration. This model is composed of an input convolutional block, 16 residual blocks and a linear layer. For DUNs, our prior over depth is uniform over the first 13 residual blocks. The last 3 residual blocks and linear layer form the output block, providing the flexibility to make predictions from activations at multiple resolutions. We use 1×1 convolutions to adapt the number of channels between earlier blocks and the output block. We use default PyTorch training hyperparameters[1] for all methods. We set per-dataset learning rate schedules. We use 5-element ensembles, as suggested by Ovadia et al. (2019), and 10 dropout samples. Mean values and standard deviations are computed across 5 independent training runs. Full details are given in Section A.2.3.

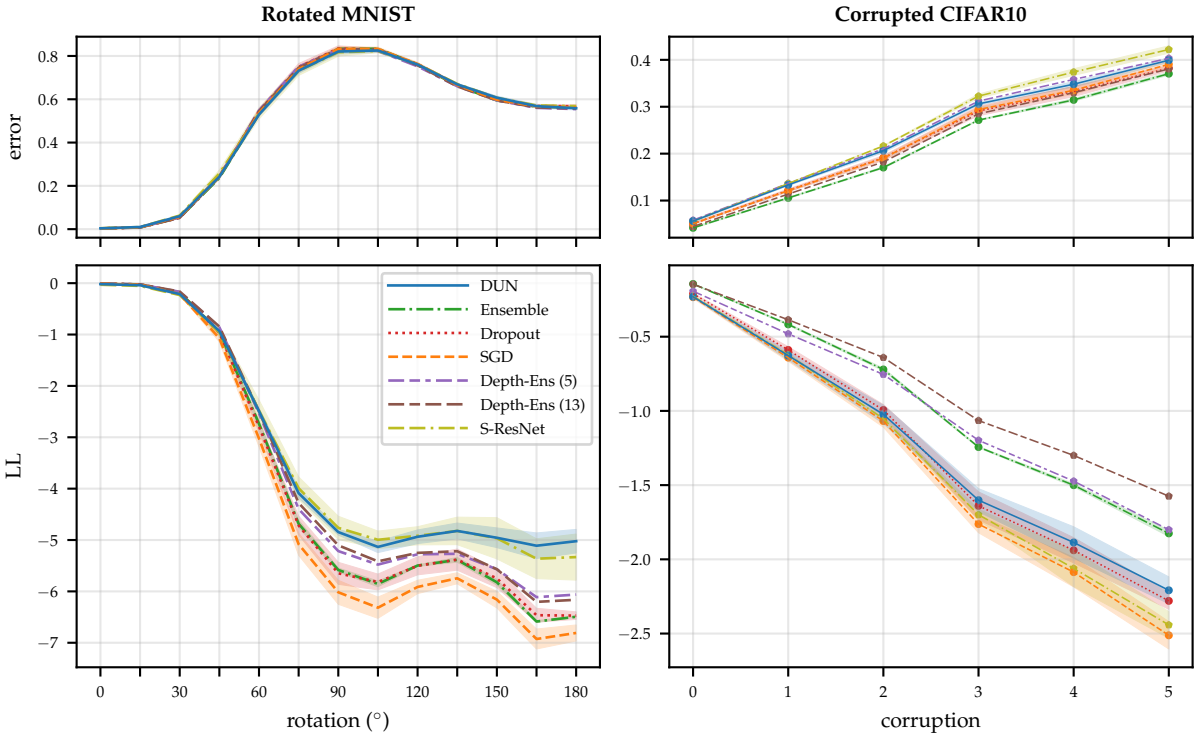


Figure 3.9: Left: error and LL for MNIST at varying degrees of rotation. Right: error and LL for CIFAR10 at varying corruption severities.

Rotated MNIST In Figure 3.9, following Ovadia et al. (2019), we train all methods on MNIST and evaluate their predictive distributions on increasingly rotated digits. Although all methods perform well on the original test set, their accuracy degrades

[1] https://github.com/pytorch/examples/blob/master/imagenet/main.py

quickly for rotations larger than 30°. Here, DUNs and Stochastic-ResNets differentiate themselves by being the least overconfident. We hypothesise that predictions based on features at diverse resolutions allow for increased disagreement. This hypothesis is supported by Figure 3.10, which shows that methods that make predictions at different resolutions do indeed have increased disagreement, as measured by the KLD between different layers' or samples' predictions.

Corrupted CIFAR In Figure 3.9, again following Ovadia et al. (2019), we train models on CIFAR10 and evaluate them on data subject to 16 different corruptions with 5 levels of intensity each (Hendrycks and Dietterich, 2019). Here, Ensembles (standard and multi-depth) significantly outperform all single network methods in terms of error and LL at all corruption levels. DUNs perform similarly to SGD and Dropout on the uncorrupted data. Despite only requiring a single forward pass for predictions, LL values reveal DUNs to be the second most robust to corruption. Interestingly, in this case, Stochastic-ResNets do not perform well—only being better than SGD.

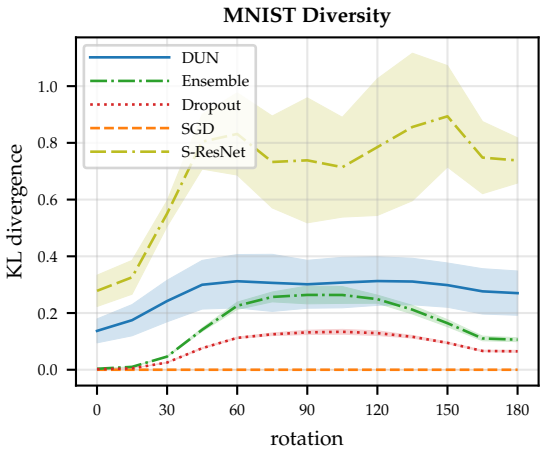


Figure 3.10: Comparison of different models' Kullback-Leibler Divergence (KLD) between predictions.

OOD Rejection In Figure 3.11, we simulate a realistic OOD rejection scenario (Filos et al., 2019) by jointly evaluating our models on an in-distribution and an OOD test set. We allow our methods to reject increasing proportions of the data based on predictive entropy before classifying the rest. All predictions on OOD samples are treated as incorrect. Ensembles are a clear best in most cases. DUNs provide a mixed performance, sometimes performing well and other times showing underconfidence: they are incapable of separating very uncertain in-distribution inputs from OOD points. We re-run DUNs using the exact posterior over depth $p(\mathrm{d} \mid \mathcal{D}; \boldsymbol{\theta})$ in (3.8), instead of $q_{\boldsymbol{\alpha}}(\mathrm{d})$. The exact posterior is computed while setting batch-norm to test mode. This also performs well in some cases while showing underconfidence in others. This inconsistency requires further investigation.

Compute Time We compare methods' performance on corrupted CIFAR10 as a function of computational budget. In general, DUNs are Pareto superior to the baselines,

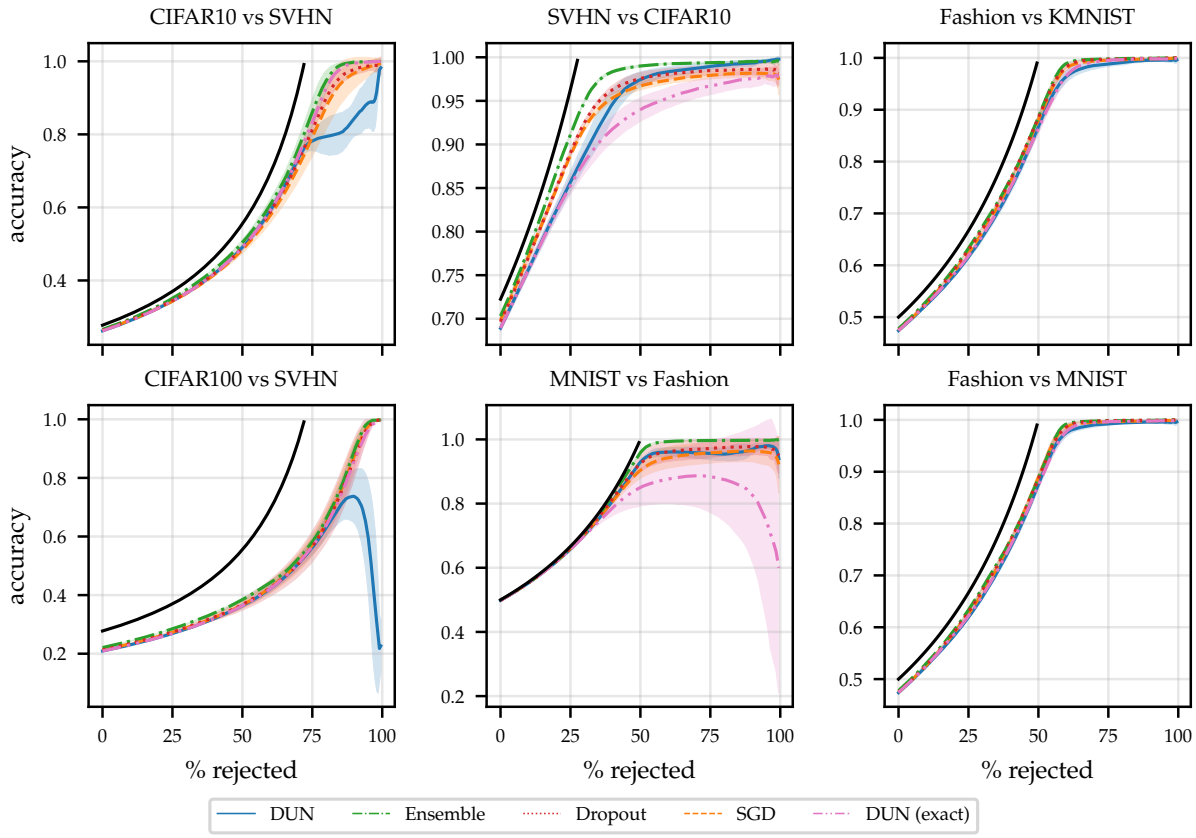


Figure 3.11: Rejection-classification plots. The black line denotes the theoretical maximum performance; all in-distribution samples are correctly classified, and OOD samples are rejected first.

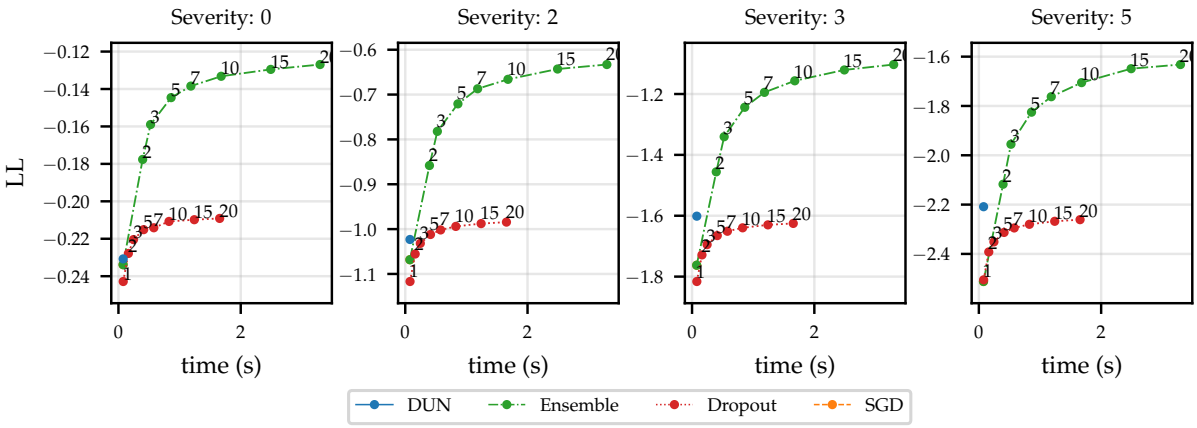


Figure 3.12: Pareto frontiers showing LL for corrupted CIFAR10 vs batch prediction time. The batch size is 256, split over 2 Nvidia P100 GPUs. Annotations show ensemble elements and Dropout samples. Note that a single-element ensemble is equivalent to SGD.

with the superiority growing larger as the severity of the corruption is increased. In the best case, for the most severe corruptions, the LL obtained by a DUN matches that of a ∼1.8 element ensemble. A single DUN forward pass is ∼1.02 times slower than a vanilla network's. On average, DUNs' computational budget matches that of ∼0.47 ensemble elements or ∼0.94 dropout samples. These values are smaller than one due to overhead, such as ensemble element loading. Thus, making predictions with DUNs is 10× faster than with five-element ensembles.

Scaling to ImageNet We train DUN (1-13) and baselines on the ILSVRC2015 dataset (Russakovsky et al., 2015). Figure 3.13 shows the performance of DUNs and various baselines for increasing corruption levels. In contrast to Figure 3.9, DUNs perform the worst. This is likely due to ResNet-50 not having sufficient capacity to perform both representation learning and make predictions at multiple depths. However, as these results do not include error bars, we cannot make strong conclusions. Nevertheless, it would be interesting to train a larger DUN, such as a ResNet-112, and compare the results.

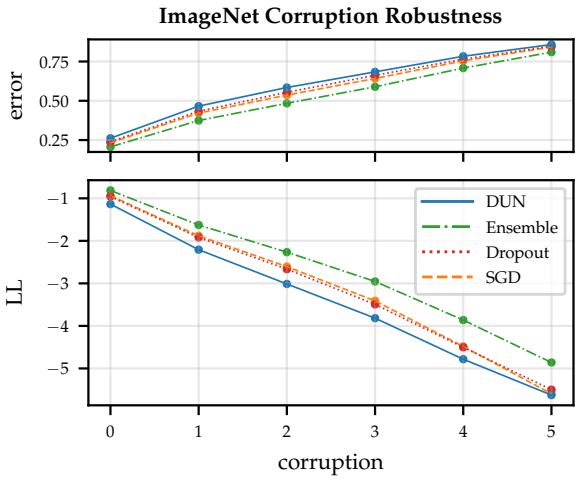


Figure 3.13: Error and LL for ImageNet at varying corruption severities.

3.3.5 DUNs for Neural Architecture Search

In this section, we briefly explore the application of DUNs to NAS.

After training a DUN, as described in Section 3.2.2, $q_{\boldsymbol{\alpha}}(\mathrm{d} = d) = \boldsymbol{\alpha}_d$ represents our confidence that the depth we should use is d. We would like to use this information to prune our network such that we reduce computational cost while maintaining performance. Recall our training objective (3.4):

$$\mathcal{L}(\boldsymbol{\alpha}, \boldsymbol{\theta}) = \sum_{n=1}^{N} \mathbb{E}_{q_{\boldsymbol{\alpha}}(\mathrm{d})} \left[\log p\left(\mathbf{y}^{(n)} \,\middle|\, \mathbf{x}^{(n)}, \mathrm{d}; \boldsymbol{\theta} \right) \right] - \mathbb{D}_{\mathrm{KL}} \left[q_{\boldsymbol{\alpha}}(\mathrm{d}) \,||\, p_{\boldsymbol{\beta}}(\mathrm{d}) \right].$$

Notice that the likelihood term scales with the dataset size, while the KLD does not. In low data regimes, where both the log-likelihood and KLD terms are of comparable scale,

we obtain a posterior with a clear maximum. We choose

$$d_{\text{opt}} = \arg\max_d \alpha_d \tag{3.10}$$

as our fixed depth. In medium-to-big data regimes, where the log-likelihood dominates our objective, we find that the values of α_i flatten out after reaching an appropriate depth. For examples of this phenomenon, compare the approximate posteriors over depth shown in Figure 3.14 and Figure 3.15. We propose two heuristics for choosing d_{opt} in this setting. The first is to use the expected value of the depth:

$$d_{\text{opt}} = \text{round}(\mathbb{E}_{q_{\boldsymbol{\alpha}}(\mathrm{d})}[\mathrm{d}]). \tag{3.11}$$

The second is to choose the depth that achieves 95% of the maximum posterior density:

$$d_{\text{opt}} = \min_i \{i : q(\mathrm{d} = i) \geq 0.95 \max_j q(\mathrm{d} = j)\}. \tag{3.12}$$

Both heuristics aim to keep the minimum number of blocks needed to explain the data well. We prune all blocks after d_{opt} by setting $q_{\boldsymbol{\alpha}}(\mathrm{d} = d_{\text{opt}}) = q_{\boldsymbol{\alpha}}(\mathrm{d} \geq d_{\text{opt}})$ and then $q_{\boldsymbol{\alpha}}(\mathrm{d} > d_{\text{opt}}) = 0$. We refer to pruned DUNs as Learnt Depth Networks (LDNs) and contrast them with (standard) Deterministic Depth Networks (DDNs) in the following experiments.

We generate a 2D training set by drawing 200 samples from a 720° rotation 2-armed spiral function with additive Gaussian noise of $\sigma = 0.15$. The test set is composed of an additional 1800 samples. Choosing a relatively small width for each hidden layer $w = 20$ to ensure the task can not be solved with a shallow model, we train fully-connected LDNs with varying maximum depths D and DDNs of all depths up to $D = 100$. Figure 3.14 shows how the depths to which LDNs assign larger probabilities match those at which DDNs perform best. Predictions from LDNs pruned to d_{opt} layers outperform DDNs at all depths. The chosen d_{opt} remains stable for increasing maximum depths up to $D \approx 50$. The same is true for test performance, showing some robustness to overfitting. After this point, training starts to become unstable. We repeat experiments 6 times and report standard deviations as error bars.

We further evaluate LDNs on MNIST, Fashion-MNIST, and SVHN. Note that the network architecture used for these experiments is different from that used for experiments on the same datasets in Section 3.3.4. It is described below. Each experiment is repeated 4 times to produce error bars. The results obtained with $D = 50$ are shown in Figure 3.15. The larger size of these datasets diminishes the effect of the prior on the ELBO. Models

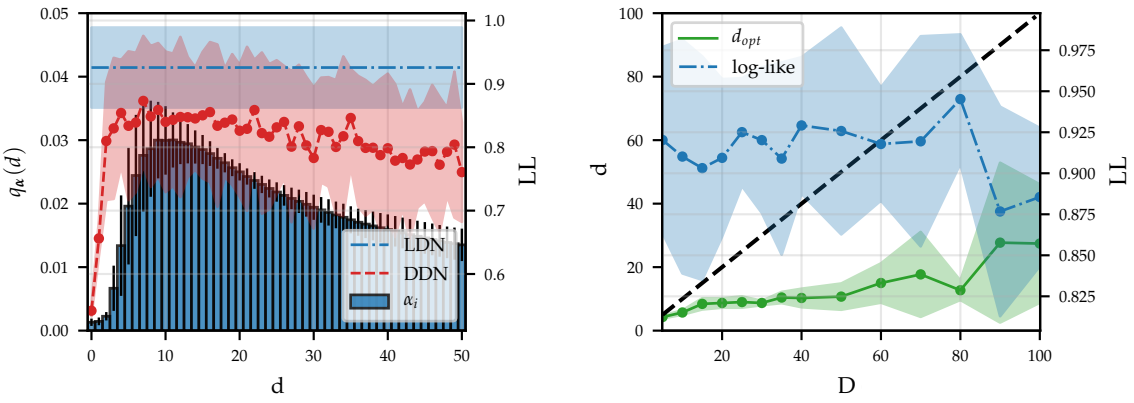


Figure 3.14: Left: posterior over depths for a LDN of $D = 50$ trained on our spirals dataset. Test log-likelihoods obtained for DDNs at every depth are overlaid with the log-likelihood value obtained with a LDN when marginalising over $d_{\text{opt}} = 9$ layers. Right: the LDN's depth, chosen using (3.10), and test performance remain stable as D increases up until $D \approx 50$.

that explain the data well obtain large probabilities, regardless of their depth. For MNIST, the probabilities assigned to each depth in our LDN grow quickly and flatten out around $d_{\text{opt}} \approx 18$. For Fashion-MNIST, depth probabilities grow more slowly. We obtain $d_{\text{opt}} \approx 28$. For SVHN, probabilities flatten out around $d_{\text{opt}} \approx 30$. These distributions and d_{opt} values correlate with dataset complexity. In most cases, LDNs achieve test log-likelihoods competitive with the best performing DDNs.

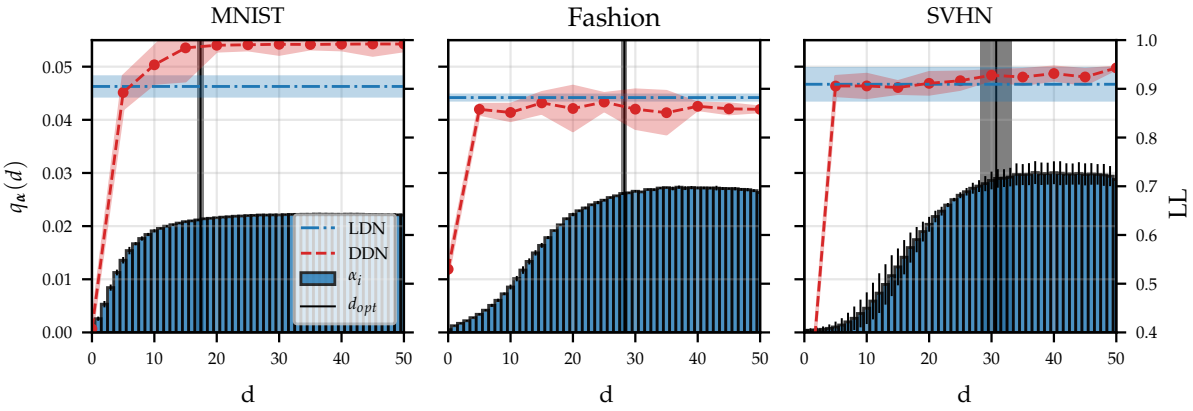


Figure 3.15: Approximate posterior over depths for LDNs of $D = 50$ trained on image datasets. Test log-likelihoods obtained for DDNs at various depths are overlaid with those from our LDNs when marginalising over the first d_{opt} layers. The depth was chosen using (3.12)

Figure 3.16 shows more detailed experiments comparing LDNs with DDNs on image datasets. The first row of the figure adds further evidence that the depth learnt by DUNs corresponds to dataset complexity. For any maximum depth, and both pruning approaches, we see that d_{opt} is smaller for MNIST than Fashion-MNIST and likewise smaller for Fashion-MNIST than SVHN. For MNIST, Fashion-MNIST, and, to a lesser extent, SVHN, the depth given by the 95^{th} percent pruning tends to saturate. On the other hand, the expected depth grows with D, making it a less suitable pruning strategy.

As shown in rows 2 to 5, for SVHN and Fashion-MNIST, 95^{th} percentile-pruned LDNs suffer no loss in predictive performance compared to expected depth-pruned and even non-pruned LDNs. They outperform DDNs. For MNIST, 95^{th} percent pruning gives results with high variance and reduced predictive performance in some cases. Here, DDNs yield better log-likelihood and accuracy results. Expected depth is more resilient in this case, matching full-depth LDNs and DDNs in terms of accuracy.

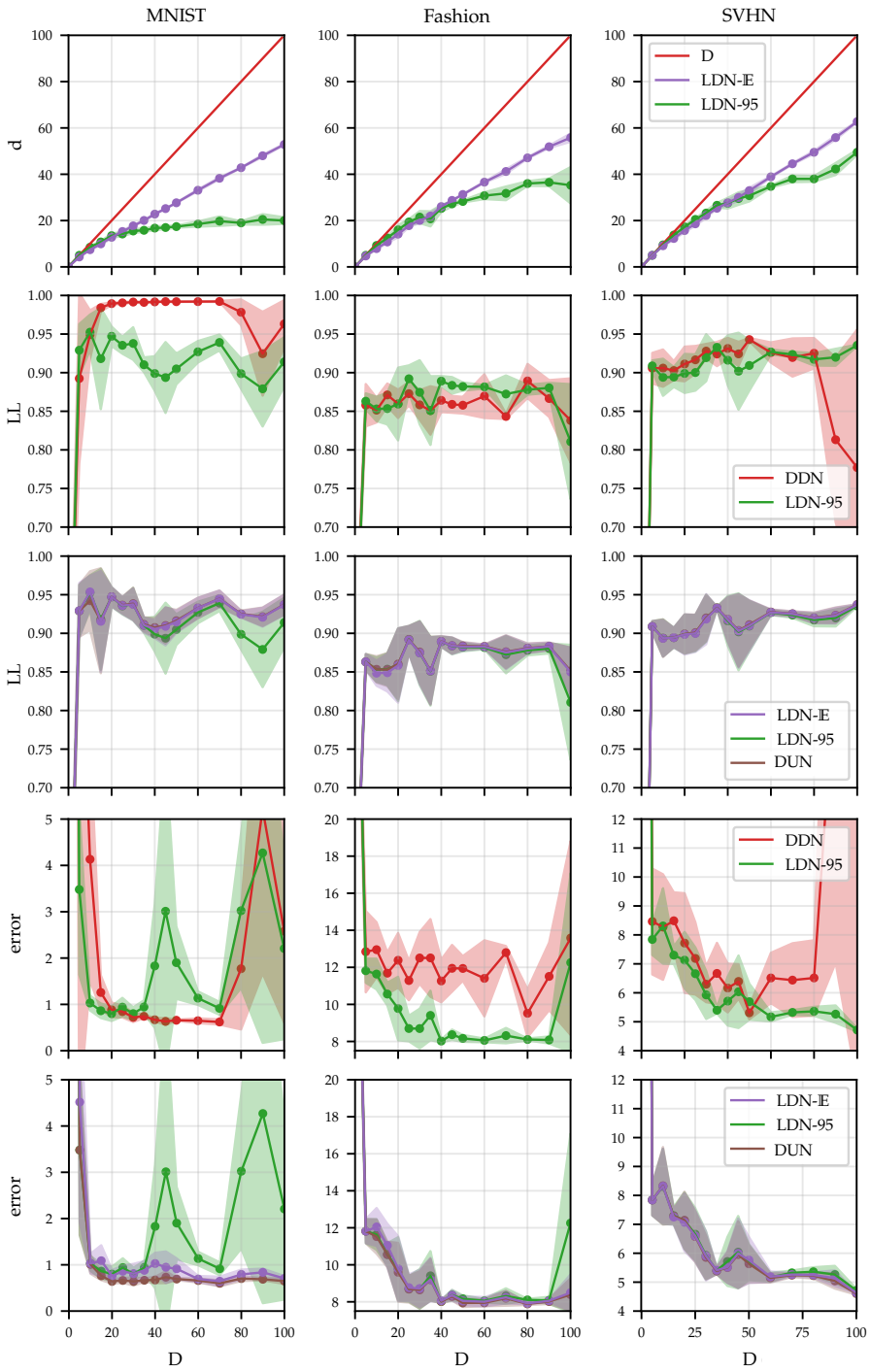


Figure 3.16: Comparisons of DDNs and LDNs using different pruning strategies and maximum depths. *LDN-95* and *LDN-*$\mathbb{E}$ refer to the pruning strategies described in (3.12) and (3.11), respectively. 1st row: comparison of d_{opt}. 2nd row: comparison of test log-likelihoods for DDNs and LDNs with 95 percent pruning. 3rd row: comparison of test log-likelihoods for LDNs pruning methods. 4th and 5th rows: as above but for test error.

3.3.6 DUNs for Active Learning

In the active learning framework, a model is initially trained on a small labelled subset of the available data, and additional unlabelled points are selected via an *acquisition function* to be labelled by an external *oracle* (e.g., a human expert) (Settles, 2009). Given a model with parameters $\boldsymbol{\theta}$ trained on training data $\mathcal{D}_{\text{train}} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$, the acquisition function $\alpha(\cdot)$ scores all unlabelled examples in the pool set $\mathcal{D}_{\text{pool}} = \{\mathbf{x}_j\}_{j=1}^{N_{\text{pool}}}$. These scores are used to select the next point $\mathbf{x}^\star$ to be labelled:

$$\mathbf{x}^\star = \underset{\mathbf{x} \in \mathcal{D}_{\text{pool}}}{\operatorname{argmax}}\ \alpha_{\text{BALD}}(\mathbf{x};\ \boldsymbol{\theta},\ \mathcal{D}_{\text{train}}). \tag{3.13}$$

Houlsby et al. (2011) propose an acquisition called Bayesian Active Learning by Disagreement (BALD),

$$\alpha_{\text{BALD}}(\mathbf{x};\ \boldsymbol{\theta},\ \mathcal{D}_{\text{train}}) = \mathbb{H}\left[p\left(\mathbf{y} \mid \mathbf{x}, \mathcal{D}_{\text{train}}\right)\right] - \mathbb{E}_{p(\boldsymbol{\theta} \mid \mathcal{D}_{\text{train}})}\left[\mathbb{H}\left[p\left(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}\right)\right]\right], \tag{3.14}$$

which selects points for which the predictions of individual parameterisations maximally disagree—i.e., where there is high uncertainty in the predictive posterior on average—but the predictions of individual parameter settings are confident. To avoid acquiring correlated points when performing batch acquisition, we implement a stochastic relaxation of BALD,

$$\mathbf{x}^\star \sim \alpha_{\text{BALDstoch}}(\mathbf{x};\ \boldsymbol{\theta},\ \mathcal{D}_{\text{train}}) = \texttt{softmax}\left(T\alpha_{\text{BALD}}(\mathbf{x};\ \boldsymbol{\theta},\ \mathcal{D}_{\text{train}})\right). \tag{3.15}$$

We use temperature $T = 10$; see Section A.2.5 for discussion of this choice. Kirsch et al. (2023) provide extensive analysis showing that such stochastic relaxations are never worse than their deterministic counterparts, and can even outperform computationally expensive batch-aware methods.

We perform experiments on eight UCI regression datasets (Hernández-Lobato and Adams, 2015). The experimental setup is described in Section A.2.5. Experiments are repeated 40 times, with the mean and standard deviations reported in the figures.

First, we investigate whether DUNs adapt their inferred depth to the size of the dataset. Figure 3.17 compares the posterior probabilities over depth for DUNs trained on datasets from the first and final steps of active learning, and illustrates that this does occur in practice.

We compare the Negative Log Likelihood (NLL) performance of DUNs to MFVI and Monte Carlo Dropout (MCDO) (Gal and Ghahramani, 2016) in Figure 3.18. The test

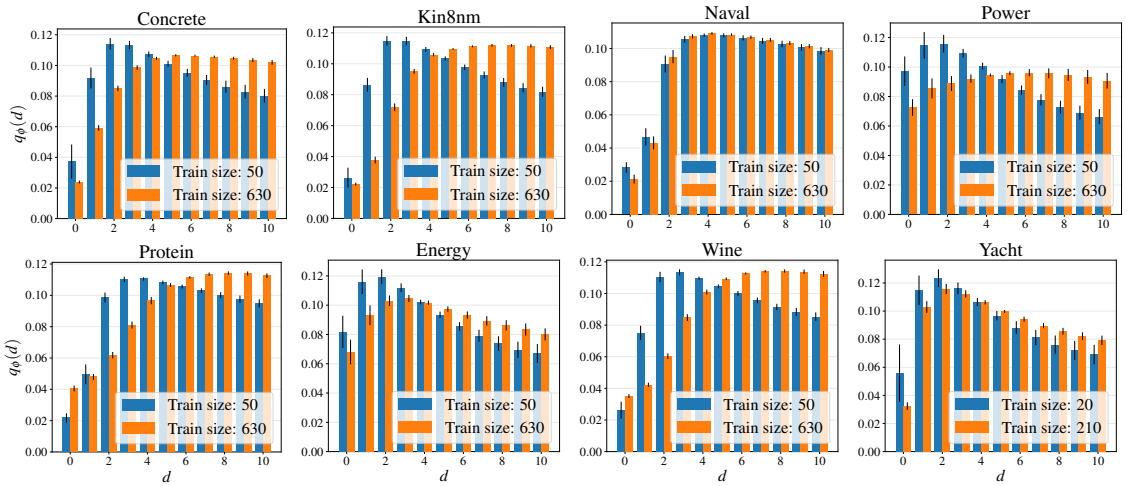


Figure 3.17: DUN depth posteriors for three UCI regression datasets, with the **smallest** and **largest** labelled datasets used during active learning.

NLL for MFVI and MCDO is evaluated using 10 MC samples. DUNs perform either better than, or comparably to, MCDO, and clearly outperforms MFVI.

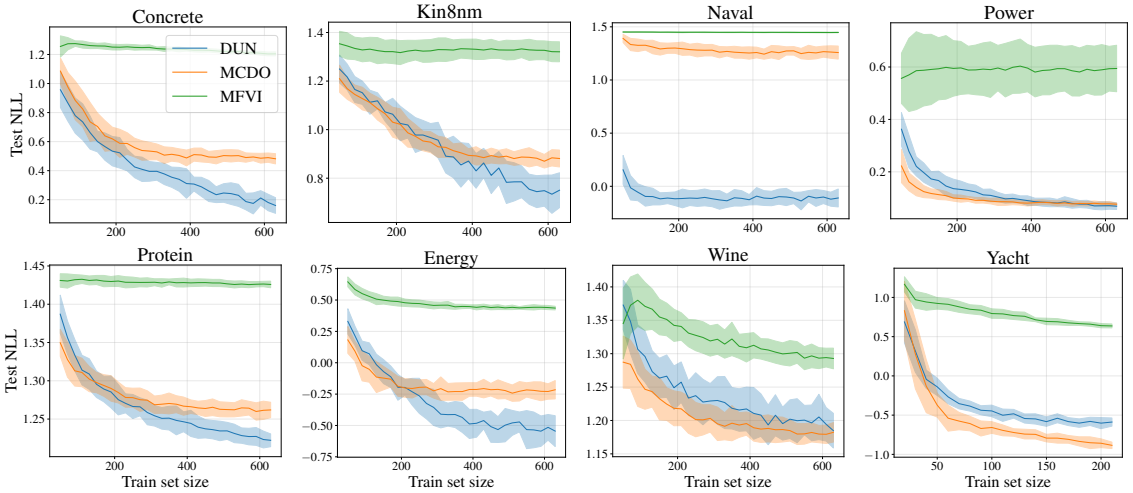


Figure 3.18: Test NLL vs. number of training points using stochastic BALD acquisition function evaluated on UCI datasets. **DUNs**, **MCDO** and **MFVI** are compared.

3.4 Related Work

Traditionally, Bayesians tackle overconfidence in deep networks by treating their weights as random variables. Through marginalisation, uncertainty in weight-space is translated to predictions. Alas, the weight posterior in Bayesian Neural Networks (BNNs) is intractable. Hamiltonian Monte Carlo (HMC) (Neal, 1995) is theoretically well-motivated method for inference in BNNs—with strong asymptotic guarantees—but requires impractically large compute resources to perform well (Izmailov et al., 2021b). The Laplace approximation

(MacKay, 1992; Ritter et al., 2018), VI (Hinton and van Camp, 1993; Graves, 2011; Blundell et al., 2015) and expectation propagation (Hernández-Lobato and Adams, 2015) have all been proposed as alternatives. More recent methods are scalable to large models (Khan et al., 2018; Osawa et al., 2019; Dusenberry et al., 2020a). Gal and Ghahramani (2016) re-interpret dropout as VI, dubbing it MC Dropout. Other stochastic regularisation techniques can also be viewed in this light (Kingma et al., 2015; Gal, 2016; Teye et al., 2018). These can be seamlessly applied to vanilla NNs. In addition to the intractability of the weight posterior, it is not clear how to place reasonable priors over NNs weights (Wenzel et al., 2020a). DUNs avoid these issues by targeting depth. BNN inference can also be performed directly in function space (Sun et al., 2019; Hafner et al., 2019; Ma et al., 2019; Wang et al., 2019). However, this requires crude approximations to the KLD between stochastic processes.

Deep ensembles provide a non-Bayesian method for uncertainty estimation in NNs that trains multiple independent networks and aggregates their predictions (Lakshminarayanan et al., 2017). Ensembling trades off strong results for high computational cost. Huang et al. (2017), Garipov et al. (2018), and Maddox et al. (2019) reduce the cost of training an ensemble by leveraging different weight configurations found in a single SGD trajectory. However, this comes at the cost of reduced predictive performance (Ashukha et al., 2020). Similarly to deep ensembles, DUNs combine the predictions from a set of deep models. However, this set stems from treating depth as a random variable. Unlike ensembles, BMA assumes the existence of a single correct model (Minka, 2000). In DUNs, uncertainty arises due to a lack of knowledge about how deep the correct model is. It is worth noting that deep ensembles can also be interpreted as approximate BMA (Wilson, 2020).

All of the above methods, except DUNs, require multiple forward passes to produce uncertainty estimates. This is problematic in low-latency settings or those in which computational resources are limited. Postels et al. (2019) use error propagation to approximate the dropout predictive posterior with a single forward pass. Although efficient, this approach shares pathologies with MC Dropout.

There is a rich literature on probabilistic inference for NN structure selection, starting with the *automatic relevance detection* prior (MacKay, 1994). Since then, several approaches have been introduced (Ghosh et al., 2019; Dikov and Bayer, 2019; Lawrence, 2001b). Perhaps the closest to our work is the *automatic depth determination* prior (Nalisnick et al., 2019a). Huang et al. (2016) stochastically drop layers as a ResNet training regularisation approach. Conversely, DUNs perform marginalisation over architectures, translating depth uncertainty into uncertainty over functional complexities.

3.5 Summary

We have recast NN depth as a random variable. This treatment allows us to optimise weights as model hyperparameters, preserving much of the simplicity of non-Bayesian NNs. Critically, both the model evidence and predictive posterior for DUNs can be evaluated with a single forward pass. Our experiments show that DUNs produce well-calibrated uncertainty estimates, performing well relative to their computational budget on uncertainty-aware tasks.

In this chapter, we have investigated uncertainty estimation in NNs by avoiding the challenges of inference in weight space and instead inferring depth. In the next chapter, we will switch gears and instead consider a method for tractable inference in weight space.

Chapter 4

Bayesian Deep Learning via Subnetwork Inference

In this chapter, we introduce *subnetwork inference*, a method for scalable Bayesian deep learning. Our contributions are:

1. In Section 4.2, we describe a general framework for scalable Bayesian deep learning in which inference is performed over only a *small subset* of the NN weights, while all other weights are kept deterministic. This allows us to use *expressive posterior approximations* that are typically intractable in large NNs. This framework is summarised in Figure 4.1.

2. After discussing the linearised Laplace algorithm in Section 4.3, we describe how to apply subnetwork inference to the linearised Laplace posterior in Section 4.4. Concretely, we first fit a MAP estimate of the full NN, and then use the linearised Laplace approximation to infer a *full-covariance Gaussian posterior* over a subnetwork.

3. In Section 4.5, we derive a subnetwork selection strategy based on the Wasserstein distance between the approximate posterior for the full network and the approximate posterior for the subnetwork. For scalability, we employ a diagonal approximation during subnetwork selection. Selecting a small subnetwork then allows us to infer weight covariances. Empirically, we find that making approximations during subnetwork selection is much less harmful to uncertainty estimation using posterior predictive than making them during inference.

4. We empirically evaluate our method on a range of benchmarks for *uncertainty calibration* and *robustness to distribution shift*, in Section 4.6. Our experiments

demonstrate that expressive subnetwork inference can outperform popular Bayesian deep learning methods that do less expressive inference over the full NN as well as deep ensembles.

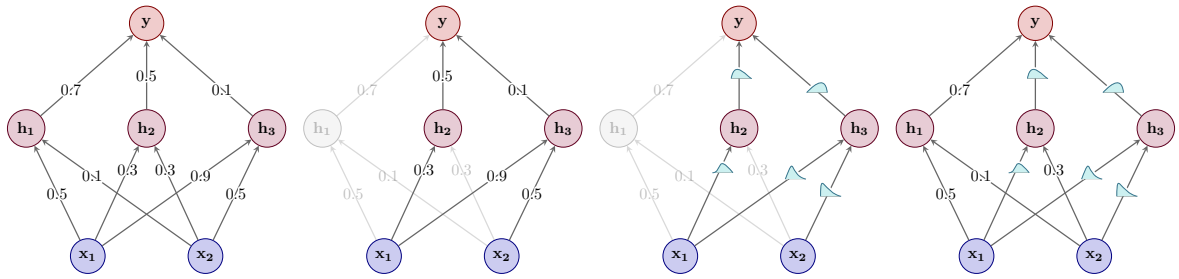
This chapter is based on the paper "Bayesian Deep Learning via Subnetwork Inference" (Daxberger et al., 2021b). This paper was written in collaboration with Erik Daxberger, Javier Antorán, Eric Nalisnick and José Miguel Hernández-Lobato. The original idea for the project was due to Erik. I was heavily involved with all aspects of the project, taking equal responsibility with Erik and Javier for conceptualisation, exploration, coding, derivations, evaluation, and presentation of the results, as well as writing the paper. Eric and Miguel provided guidance and high-level input throughout the projects, with Eric also being very involved with the initial ideation and several derivations.

4.1 Motivation

A critical shortcoming of deep NNs is that they tend to be poorly calibrated and overconfident in their predictions, especially when there is a shift between the train and test data distributions (Nguyen et al., 2015; Guo et al., 2017). To reliably inform decision making, NNs need to robustly quantify the *uncertainty* in their predictions (Bhatt et al., 2021). This is especially important for safety-critical applications such as healthcare or autonomous driving (Amodei et al., 2016).

Bayesian modelling (Bishop, 2006; Ghahramani, 2015) presents a principled way to capture uncertainty via the posterior distribution over model parameters. Unfortunately, exact posterior inference is intractable in NNs. Despite continued interest in the field of Bayesian deep learning (see Section 2.1.2), existing methods invoke unrealistic assumptions to scale to NNs with large numbers of weights. For example, Osawa et al. (2019) scale Bayesian Neural Networks (BNNs) to ImageNet classification, but rely on the strong mean-field approximation. Such approximations severely limit the expressiveness of the inferred posterior and thus deteriorate the quality of the induced uncertainty estimates (Ovadia et al., 2019; Fort et al., 2019; Foong et al., 2020).

Perhaps these unrealistic inference approximations can be avoided. Due to the heavy overparameterisation of NNs, their accuracy is well-preserved by a small subnetwork (Cheng et al., 2017). Moreover, doing inference over a low-dimensional subspace of the weights can result in accurate uncertainty quantification (Izmailov et al., 2019). This prompts the following question: *Can a full NN's model uncertainty be well-preserved by a small subnetwork?* In this chapter, we demonstrate that the posterior predictive distribution of a full network *can* be well represented by that of a subnetwork.



(a) Point Estimation **(b)** Subnet. Selection **(c)** Bayesian Inference **(d)** Prediction

Figure 4.1: Schematic illustration of our proposed approach. (a) We train a neural network using standard techniques to obtain a point estimate of the weights. (b) We identify a small subset of the weights. (c) We estimate a posterior distribution over the selected subnetwork via Bayesian inference techniques. (d) We make predictions using the full network with a mix of Bayesian and deterministic weights.

4.2 Subnetwork Posterior Approximation

Let $\mathbf{w} \in \mathbb{R}^D$ be the D-dimensional vector of all neural network weights (i.e., the concatenation and flattening of all layers' weight matrices). BNNs aim to capture *model uncertainty*, i.e., uncertainty about the choice of weights $\mathbf{w}$ arising due to multiple plausible explanations of the training data $\mathcal{D} = \{\mathbf{y}, \mathbf{X}\}$. Here, $\mathbf{y} \in \mathbb{R}^O$ is the output variable (e.g., classification label) and $\mathbf{X} \in \mathbb{R}^{N \times I}$ is the feature matrix. First, a prior distribution $p(\mathbf{w})$ is specified over the BNN's weights $\mathbf{w}$. We then wish to infer their full *posterior distribution*

$$p(\mathbf{w} \mid \mathcal{D}) = p(\mathbf{w} \mid \mathbf{y}, \mathbf{X}) \propto p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) p(\mathbf{w}). \tag{4.1}$$

Finally, predictions for new data points $\mathbf{X}^*$ are made through marginalisation of the posterior:

$$p(\mathbf{y}^* \mid \mathbf{X}^*, \mathcal{D}) = \int_{\mathbf{w}} p(\mathbf{y}^* \mid \mathbf{X}^*, \mathbf{w}) p(\mathbf{w} \mid \mathcal{D}) \mathrm{d}\mathbf{w}. \tag{4.2}$$

This *posterior predictive* distribution translates uncertainty in weights to uncertainty in predictions. Unfortunately, due to the non-linearity of NNs, it is intractable to infer the exact posterior distribution $p(\mathbf{w} \mid \mathcal{D})$. It is even computationally challenging to faithfully *approximate* the posterior due to the high dimensionality of $\mathbf{w}$. Thus, crude posterior approximations such as complete factorization, i.e., $p(\mathbf{w} \mid \mathcal{D}) \approx \prod_{d=1}^{D} q(\mathrm{w}_d)$ where w_d is the d^{th} weight in $\mathbf{w}$, are commonly employed (Hernández-Lobato and Adams, 2015;

Blundell et al., 2015; Khan et al., 2018; Osawa et al., 2019). However, it has been shown that such an approximation suffers from severe pathologies (Foong et al., 2019, 2020).

In this chapter, we question the widespread implicit assumption that an expressive posterior approximation must include *all* D of the model weights to make predictions with well-calibrated uncertainty estimates. Instead, we try to perform inference only over a *small subset* of $S \ll D$ of the weights. The following arguments motivate this approach:

1. **Overparameterisation:** Maddox et al. (2020) have shown that, in the neighbourhood of local optima, there are many directions that leave the NN's predictions unchanged. Moreover, NNs can be heavily pruned without sacrificing test-set accuracy (Frankle and Carbin, 2019). This suggests that the majority of a NN's predictive power can be isolated to a small subnetwork.

2. **Inference over submodels:** Previous work[1] has provided evidence that inference can be effective even when not performed on the full parameter space. Examples include Izmailov et al. (2019) and Snoek et al. (2015), who perform inference over low-dimensional projections of the weights, and only the last layer of a NN, respectively.

We therefore combine these two ideas and make the following two-step approximation of the posterior in (4.1):

$$p(\mathbf{w} \mid \mathcal{D}) = p(\mathbf{w}_S \mid \mathcal{D}, \mathbf{w}_R) p(\mathbf{w}_R \mid \mathcal{D}) \tag{4.3}$$

$$\approx p(\mathbf{w}_S \mid \mathcal{D}, \mathbf{w}_R) \prod_r \delta(\mathrm{w}_r - \widehat{\mathrm{w}}_r) \tag{4.4}$$

$$\approx q(\mathbf{w}_S) \prod_r \delta(\mathrm{w}_r - \widehat{\mathrm{w}}_r) = q_S(\mathbf{w}). \tag{4.5}$$

We first decompose the full NN posterior $p(\mathbf{w} \mid \mathcal{D})$ into a posterior $p(\mathbf{w}_S \mid \mathcal{D}, \mathbf{w}_R)$ over the subnetwork $\mathbf{w}_S \in \mathbb{R}^S$ and a posterior $p(\mathbf{w}_R \mid \mathcal{D})$ over the remaining weights $\mathbf{w}_R \in \mathbb{R}^{D-S}$. The first approximation (4.4) replaces the later with Dirac delta functions $\delta(\mathrm{w}_r - \widehat{\mathrm{w}}_r)$ to keep w_r at fixed values $\widehat{\mathrm{w}}_r$. Since posterior inference over the subnetwork is still intractable, (4.5) further approximates $p(\mathbf{w}_S \mid \mathcal{D}, \mathbf{w}_R)$ by $q(\mathbf{w}_S)$. However, importantly, if the subnetwork is much smaller than the full network, we can afford to make $q(\mathbf{w}_S)$ *more expressive* than would otherwise be possible. We hypothesise that being able to capture rich dependencies across the weights within the subnetwork will provide better results than crude approximations applied to the full set of weights.

[1] See Section 4.8 for a more thorough discussion of related work.

Relationship to Weight Pruning Methods. Note that the posterior approximation in (4.5) can be viewed as *pruning the variances* of the weights $\{\mathrm{w}_r\}_r$ to zero. This is in contrast to weight pruning methods[2] that set the *weights* themselves to zero. I.e., weight pruning methods can be viewed as removing *weights* to preserve the predictive *mean* (i.e., to retain *accuracy* close to the full model). In contrast, subnetwork inference can be viewed as removing just the *variances* of certain weights—while keeping their means—to preserve the predictive *uncertainty* (e.g., to retain *calibration* close to the full model). Thus, they are complementary approaches. Importantly, by not pruning weights, subnetwork inference retains the *full predictive power* of the full NN to retain its predictive accuracy.

4.3 Background: Linearised Laplace

In this chapter, we satisfy (4.5) by approximating the posterior distribution over the weights with the *linearised Laplace approximation* (MacKay, 1992). This is an inference technique that has recently been shown to perform strongly (Foong et al., 2019; Immer et al., 2021b) and can be applied *post-hoc* to pre-trained models. We now describe it in a general setting.

We denote our NN function as $\boldsymbol{f} : \mathbb{R}^I \to \mathbb{R}^O$. We begin by defining a prior over our NN's weights, which we choose to be a fully factorised Gaussian $p(\mathbf{w}) = \mathcal{N}\left(\mathbf{w} \,\middle|\, \mathbf{0},\, \lambda^{-1} \cdot \boldsymbol{I}\right)$. We find a local optimum of the posterior, also known as a MAP setting of the weights:

$$\widehat{\mathbf{w}} = \arg\max_{\mathbf{w}} \left[\log p(\mathbf{y} \mid \mathbf{X},\, \mathbf{w}) + \log p(\mathbf{w})\right]. \tag{4.6}$$

The posterior is then approximated with a second-order Taylor expansion around the MAP estimate:

$$\log p(\mathbf{w} \mid \mathcal{D}) \approx \log p(\widehat{\mathbf{w}} \mid \mathcal{D}) - \frac{1}{2}(\mathbf{w} - \widehat{\mathbf{w}})^\top \mathbf{H} (\mathbf{w} - \widehat{\mathbf{w}}) \tag{4.7}$$

where $\mathbf{H} \in \mathbb{R}^{D \times D}$ is the Hessian of the negative log-posterior density with respect to the network weights $\mathbf{w}$:

$$\mathbf{H} = N \cdot \mathbb{E}_{\mathcal{D}} \left[-\frac{\partial^2}{\partial \mathbf{w}^2} \log p(\mathbf{y} \mid \mathbf{X},\, \mathbf{w}) \right] + \lambda \cdot \boldsymbol{I}\,. \tag{4.8}$$

[2] See Cheng et al. (2017) for a review.

Thus, the approximate posterior takes the form of a full-covariance Gaussian with covariance matrix $\mathbf{H}^{-1}$:

$$p(\mathbf{w} \mid \mathcal{D}) \approx q(\mathbf{w}) = \mathcal{N}\left(\mathbf{w} \,\middle|\, \widehat{\mathbf{w}}, \mathbf{H}^{-1}\right). \tag{4.9}$$

In practise, the Hessian $\mathbf{H}$ is commonly replaced with the Generalized Gauss-Newton (GGN) matrix (Martens and Sutskever, 2011; Martens, 2020, 2016):

$$\widetilde{\mathbf{H}} = \sum_{n=1}^{N} \mathbf{J}_n^\top \mathbf{H}_n \mathbf{J}_n + \lambda \cdot \boldsymbol{I} \;\in \mathbb{R}^{D \times D}. \tag{4.10}$$

Where,

$$\mathbf{J}_n = \frac{\partial \boldsymbol{f}(\mathbf{x}_n, \mathbf{w})}{\partial \mathbf{w}} \in \mathbb{R}^{O \times D} \tag{4.11}$$

is the Jacobian of the model outputs $\boldsymbol{f}(\mathbf{x}_n, \mathbf{w}) \in \mathbb{R}^O$ with respect to $\mathbf{w}$, and

$$\mathbf{H}_n = -\frac{\partial^2 \log p\left(\mathbf{y} \mid \boldsymbol{f}(\mathbf{x}_n, \mathbf{w})\right)}{\partial^2 \boldsymbol{f}(\mathbf{x}_n, \mathbf{w})} \in \mathbb{R}^{O \times O} \tag{4.12}$$

is the Hessian of the negative Log Likelihood (LL) with respect to the model outputs.

Interestingly, when using a Gaussian likelihood, the Gaussian with a GGN precision matrix corresponds to the *true* posterior distribution when the NN is approximated with a first-order Taylor expansion around $\widehat{\mathbf{w}}$ (Khan et al., 2019; Immer et al., 2021b). The *locally linearised* function is

$$\boldsymbol{f}_{\text{lin}}(\mathbf{x}, \mathbf{w}) = \boldsymbol{f}(\mathbf{x}, \widehat{\mathbf{w}}) + \widehat{\mathbf{J}}(\mathbf{x})(\mathbf{w} - \widehat{\mathbf{w}}), \tag{4.13}$$

where

$$\widehat{\mathbf{J}}(\mathbf{x}) = \frac{\partial \boldsymbol{f}(\mathbf{x}, \widehat{\mathbf{w}})}{\partial \widehat{\mathbf{w}}} \in \mathbb{R}^{O \times D}. \tag{4.14}$$

This turns the underlying probabilistic model from a BNN into a Generalized Linear Model (GLM), where the Jacobian $\widehat{\mathbf{J}}(\mathbf{x})$ acts as a basis function expansion. Making predictions with the GLM $\boldsymbol{f}_{\text{lin}}$ has been found to outperform the corresponding BNN $\boldsymbol{f}$ with the GGN-Laplace posterior (Lawrence, 2001a; Foong et al., 2019; Immer et al., 2021b). Additionally, the equivalence between a GLM and a linearised BNN will help us to derive a subnetwork selection strategy in Section 4.5.

The resulting posterior predictive distribution is

$$p(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D}) = \int p(\mathbf{y}^* \mid \boldsymbol{f}_{\text{lin}}(\mathbf{x}^*, \mathbf{w}))p(\mathbf{w} \mid \mathcal{D})\mathrm{d}\mathbf{w}. \tag{4.15}$$

For regression—with a Gaussian likelihood $p(\mathbf{y}^* \mid \boldsymbol{f}_{\text{lin}}(\mathbf{x}^*, \mathbf{w})) = \mathcal{N}(\mathbf{y}^* \mid \boldsymbol{f}_{\text{lin}}(\mathbf{x}^*, \mathbf{w}), \sigma^2)$—our approximate distribution becomes exact $q(\mathbf{w}) = p(\mathbf{w} \mid \mathcal{D}) = \mathcal{N}\left(\mathbf{w} \,\middle|\, \widehat{\mathbf{w}}, \widetilde{\mathbf{H}}^{-1}\right)$. We obtain the closed-form predictive

$$p(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D}) = \mathcal{N}\left(\mathbf{y}^* \,\middle|\, \boldsymbol{f}(\mathbf{x}^*, \widehat{\mathbf{w}}), \boldsymbol{\Sigma}(\mathbf{x}^*) + \sigma^2 \cdot \boldsymbol{I}\right), \tag{4.16}$$

where $\boldsymbol{\Sigma}(\mathbf{x}^*) = \widehat{\mathbf{J}}(\mathbf{x}^*)^\top \widetilde{\mathbf{H}}^{-1} \widehat{\mathbf{J}}(\mathbf{x}^*)$. For classification—with a categorical likelihood $p(\mathbf{y}^* \mid \boldsymbol{f}_{\text{lin}}(\mathbf{x}^*, \mathbf{w})) = \text{Cat}(\mathbf{y}^* \mid \phi(\boldsymbol{f}_{\text{lin}}(\mathbf{x}^*, \mathbf{w}))$—the posterior is strictly convex. This justifies our choice of approximating a strictly convex distribution by a Gaussian that is also strictly convex. Here, $\phi(\cdot)$ refers to the softmax function. The predictive integral has no analytical solution. Instead we leverage the probit approximation (Gibbs, 1998; Bishop, 2006):

$$p(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D}) \approx \text{Cat}\left(\mathbf{y}^* \,\middle|\, \phi\left(\frac{\boldsymbol{f}(\mathbf{x}^*, \widehat{\mathbf{w}})}{\sqrt{1 + \frac{\pi}{8}\text{diag}(\Sigma(\mathbf{x}^*))}}\right)\right). \tag{4.17}$$

These closed-form expressions are attractive since they result in the predictive mean and classification boundaries being exactly equal to those of the MAP estimated NN.

Unfortunately, storing the full $D \times D$ covariance matrix over the weight space of a modern NN (i.e., with very large D) is computationally intractable. There have been efforts to develop cheaper approximations to this object, such as only storing diagonal (Denker and LeCun, 1990) or block diagonal (Ritter et al., 2018; Immer et al., 2021b) entries, but these, like any approximation, come at the cost of reduced uncertainty calibration in the predictive posterior.

4.4 Linearised Laplace Subnetwork Inference

We outline the following procedure for scaling the linearised Laplace approximation to large neural network models within the framework of subnetwork inference.

Step #1: Point Estimation, Figure 4.1 (a). Train a neural network to obtain a point estimate of the weights, denoted $\widehat{\mathbf{w}}$. This can be done using stochastic gradient-

based optimisation methods (Goodfellow et al., 2016). Alternatively, we could make use of a pre-trained model.

Step #2: Subnetwork Selection, Figure 4.1 (b). Identify a small subnetwork $\mathbf{w}_S \in \mathbb{R}^S, S \ll D$. Ideally, we would like to find the subnetwork that produces a predictive posterior 'closest' to the full network's predictive distribution. Regrettably, reasoning in the space of functions directly is challenging (Burt et al., 2021). Instead, in Section 4.5, we describe a strategy that minimises the Wasserstein distance between the sub- and full-network's weight posteriors.

Step #3: Bayesian Inference, Figure 4.1 (c). Use the GGN-Laplace approximation to infer a full-covariance Gaussian posterior over the subnetwork's weights $\mathbf{w}_S \in \mathbb{R}^S$:

$$p(\mathbf{w}_S \,|\, \mathcal{D},\, \mathbf{w}_R) \approx q(\mathbf{w}_S) = \mathcal{N}\Big(\mathbf{w}_S \,\Big|\, \widehat{\mathbf{w}}_S,\, \widetilde{\mathbf{H}}_S^{-1}\Big) \tag{4.18}$$

where $\widetilde{\mathbf{H}}_S \in \mathbb{R}^{S \times S}$ is the GGN with respect to the weights $\mathbf{w}_S$:

$$\widetilde{\mathbf{H}}_S = \sum_{n=1}^{N} \mathbf{J}_{Sn}^\top \mathbf{H}_n \mathbf{J}_{Sn} + \lambda_S \cdot \boldsymbol{I}. \tag{4.19}$$

Here, $\mathbf{J}_{Sn} = \partial \boldsymbol{f}(\mathbf{x}_n, \mathbf{w}_S)/\partial \mathbf{w}_S \in \mathbb{R}^{O \times S}$ is the Jacobian with respect to $\mathbf{w}_S$. $\mathbf{H}_n$ is defined as in Section 4.2. In order to best preserve the magnitude of the predictive variance, we update our prior precision to be $\lambda_S = \lambda \cdot {}^S\!/_D$ (see Section B.1 for more details). All weights not belonging to the chosen subnetwork are fixed at their MAP values. Note that this whole procedure (i.e., Steps #1–#3) is a perfectly valid mixed inference strategy: We perform full Laplace inference over the selected subnetwork and MAP inference over all remaining weights. The resulting approximate posterior (4.5) is

$$q_S(\mathbf{w}) \overset{(4.18)}{=} \mathcal{N}\Big(\mathbf{w}_S \,\Big|\, \widehat{\mathbf{w}}_S,\, \widetilde{\mathbf{H}}_S^{-1}\Big) \prod_r \delta(\mathrm{w}_r - \widehat{\mathrm{w}}_r). \tag{4.20}$$

Given a sufficiently small subnetwork $\mathbf{w}_S$, it is feasible to store and invert $\widetilde{\mathbf{H}}_S$. In particular, naively storing and inverting the *full* GGN $\widetilde{\mathbf{H}}$ scales as $\mathcal{O}(D^2)$ and $\mathcal{O}(D^3)$, respectively. Using the subnetwork GGN $\widetilde{\mathbf{H}}_S$ instead reduces this burden to $\mathcal{O}(S^2)$ and $\mathcal{O}(S^3)$, respectively. In our experiments, $S \ll D$ with our subnetworks representing less than 1% of the total weights. Note that quadratic/cubic scaling in S is unavoidable if we are to capture weight correlations.

Step #4: Prediction, Figure 4.1 (d). Perform a local linearisation of the NN (see Section 4.3) while fixing $\mathbf{w}_r$ to $\widehat{\mathbf{w}}_r$:

$$\boldsymbol{f}_{\text{lin}}(\mathbf{x}, \mathbf{w}_S) = \boldsymbol{f}(\mathbf{x}, \widehat{\mathbf{w}}) + \widehat{\mathbf{J}}_S(\mathbf{x})(\mathbf{w}_S - \widehat{\mathbf{w}}_S) \ , \tag{4.21}$$

where $\widehat{\mathbf{J}}_S(\mathbf{x}) = \partial \boldsymbol{f}(\mathbf{x}, \widehat{\mathbf{w}}_S)/\partial \widehat{\mathbf{w}}_S \in \mathbb{R}^{O \times S}$. Following (4.16) and (4.17), the corresponding predictive distributions are

$$p\left(\mathbf{y}^* \,|\, \mathbf{x}^*, \mathcal{D}\right) = \mathcal{N}\left(\mathbf{y}^* \,\middle|\, \boldsymbol{f}(\mathbf{x}^*, \widehat{\mathbf{w}}), \Sigma_S(\mathbf{x}^*) + \sigma^2 \cdot \boldsymbol{I}\right) \tag{4.22}$$

for regression and

$$p\left(\mathbf{y}^* \,|\, \mathbf{x}^*, \mathcal{D}\right) \approx \text{softmax}\left(\frac{\boldsymbol{f}(\mathbf{x}^*, \widehat{\mathbf{w}})}{\sqrt{1 + \frac{\pi}{8}\text{diag}(\Sigma_S(\mathbf{x}^*))}}\right) \tag{4.23}$$

for classification, where $\Sigma(\mathbf{x}^*)$ in (4.16) and (4.17) is substituted with $\Sigma_S(\mathbf{x}^*) = \widehat{\mathbf{J}}_S(\mathbf{x}^*)^T \widetilde{\mathbf{H}}_S^{-1} \widehat{\mathbf{J}}_S(\mathbf{x}^*)$.

4.5 Subnetwork Selection

Ideally, we would like to choose a subnetwork such that the induced predictive posterior distribution is as close as possible to the predictive posterior provided by inference over the full network (4.15). This discrepancy between stochastic processes is often quantified through the functional Kullback-Leibler Divergence (KLD) (Sun et al., 2019; Burt et al., 2021):

$$\sup_{n \in \mathbb{N}, \mathbf{X}^* \in \mathcal{X}^n} \mathbb{D}_{\text{KL}}\left[p_S\left(\mathbf{y}^* \,|\, \mathbf{X}^*, \mathcal{D}\right) || \, p\left(\mathbf{y}^* \,|\, \mathbf{X}^*, \mathcal{D}\right)\right], \tag{4.24}$$

where p_S denotes the subnetwork predictive posterior and $\mathcal{X}^n$ denotes a finite measurement set of n elements. Regrettably, reasoning directly in function space is a difficult task (Nalisnick and Smyth, 2018; Pearce et al., 2019; Sun et al., 2019; Antorán et al., 2020; Nalisnick et al., 2021; Burt et al., 2021). Instead, we focus our attention on weight space.

In weight space, our aim is to minimise the discrepancy between the exact posterior over the full network (4.1) and the subnetwork approximate posterior (4.5). This provides two challenges. Firstly, computing the exact posterior distribution remains intractable. Secondly, common discrepancies, like the KLD or the Hellinger distance, are not useful

due to the Dirac delta distributions found in (4.5), which result in infinite penalties that do not allow us to distinguish between *good* and *bad* choices of the subnetwork.

To solve the first issue, we again resort to local linearisation, introduced in Section 4.3. The true posterior for the linearised model is Gaussian or approximately Gaussian[3]:

$$p(\mathbf{w} \mid \mathcal{D}) \simeq \mathcal{N}\left(\mathbf{w} \,\middle|\, \widehat{\mathbf{w}}, \widetilde{\mathbf{H}}^{-1}\right). \tag{4.25}$$

We solve the second issue by choosing the squared 2-Wasserstein distance, which for two Gaussian distributions $\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $\mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$, has the following closed-form expression (Givens et al., 1984)[4]:

$$W_2\left(\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1), \mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)\right)^2 = \|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\|_2^2 + \operatorname{Tr}\left(\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2 - 2\left(\boldsymbol{\Sigma}_2^{1/2}\boldsymbol{\Sigma}_1\boldsymbol{\Sigma}_2^{1/2}\right)^{1/2}\right). \tag{4.26}$$

In this case, both distributions have the same mean: $\boldsymbol{\mu}_1 = \boldsymbol{\mu}_2 = \widehat{\mathbf{w}}$. The true posterior's covariance matrix is the inverse GGN matrix, i.e., $\boldsymbol{\Sigma}_1 = \widetilde{\mathbf{H}}^{-1}$. For the approximate posterior $\boldsymbol{\Sigma}_2 = \widetilde{\mathbf{H}}_{S+}^{-1}$, which is equal to $\widetilde{\mathbf{H}}_S^{-1}$ (the inverse GGN matrix of the subnetwork) padded with zeros at the positions corresponding to point estimated weights $\mathbf{w}_r$, matching the shape of $\widetilde{\mathbf{H}}^{-1}$. Alternatively, but equivalently, we can define $\widetilde{\mathbf{H}}_{S+}^{-1} = \mathbf{M}_S \odot \widetilde{\mathbf{H}}^{-1}$, where $\odot$ is the Hadamard product, and $\mathbf{M}_S$ is a mask matrix with zeros in the rows and columns corresponding to $\mathbf{w}_r$, i.e., the rows and columns corresponding to weights not included in the subnetwork. Thus, for the case of a full covariance Gaussian (4.25) and a product of a full covariance Gaussian with Dirac deltas (4.20), this metric takes the following form:

$$\begin{aligned}
&W_2\left(p(\mathbf{w} \mid \mathcal{D}),\, q_S(\mathbf{w})\right)^2 && (4.27)\\
&= W_2\left(\mathcal{N}\left(\widehat{\mathbf{w}}, \widetilde{\mathbf{H}}^{-1}\right), \mathcal{N}\left(\widehat{\mathbf{w}}, \widetilde{\mathbf{H}}_{S+}^{-1}\right)\right)^2 && (4.28)\\
&= \cancel{\|\widehat{\mathbf{w}} - \widehat{\mathbf{w}}\|_2^2} + \operatorname{Tr}\left(\widetilde{\mathbf{H}}^{-1} + \widetilde{\mathbf{H}}_{S+}^{-1} - 2\left(\widetilde{\mathbf{H}}_{S+}^{-1/2}\widetilde{\mathbf{H}}^{-1}\widetilde{\mathbf{H}}_{S+}^{-1/2}\right)^{1/2}\right) && (4.29)\\
&= \operatorname{Tr}\left(\widetilde{\mathbf{H}}^{-1} + \widetilde{\mathbf{H}}_{S+}^{-1} - 2\left(\widetilde{\mathbf{H}}_{S+}^{-1/2}\widetilde{\mathbf{H}}^{-1}\widetilde{\mathbf{H}}_{S+}^{-1/2}\right)^{1/2}\right). && (4.30)
\end{aligned}$$

Finding the subset $\mathbf{w}_S \in \mathbb{R}^S$ of size S that minimises (4.30) would be combinatorially difficult, as the contribution of each weight depends on every other weight. To address this issue, we make an independence assumption among weights, i.e., $\widetilde{\mathbf{H}}^{-1} = \operatorname{diag}(\sigma_1^2, \ldots, \sigma_D^2)$,

[3]When not making predictions with the linearised model, the Gaussian posterior would represent a crude approximation.

[4]This also holds for our case of a degenerate Gaussian with singular covariance matrix.

resulting in the simplified objective

$$W_2\left(p\left(\mathbf{w} \mid \mathcal{D}\right),\, q_S(\mathbf{w})\right)^2 \tag{4.31}$$

$$\approx \mathrm{Tr}\left(\widetilde{\mathbf{H}}^{-1}\right) + \mathrm{Tr}\left(\widetilde{\mathbf{H}}_{S+}^{-1}\right) - 2\mathrm{Tr}\left(\widetilde{\mathbf{H}}^{-1/2}\widetilde{\mathbf{H}}_{S+}^{-1/2}\right) \tag{4.32}$$

$$= \sum_{d=1}^{D} \sigma_d^2 + m_d\sigma_d^2 - 2m_d\sigma_d^2 \tag{4.33}$$

$$= \sum_{d=1}^{D} \sigma_d^2(1 - m_d)\ , \tag{4.34}$$

where m_d is the d^{th} element of $\mathrm{diag}(\mathbf{M}_S)$, i.e., $m_d = 1$ if $\mathrm{w}_d \in \mathbf{w}_S$ and 0 otherwise. The objective (4.31) is trivially minimised by a subnetwork containing the S weights with the highest variances. This is related to common magnitude-based weight pruning methods (Cheng et al., 2017). The main difference is that our selection strategy involves weight *variances* rather than *magnitudes* as we target predictive uncertainty rather than accuracy.

In practice, even computing the marginal variances (i.e., the diagonal of $\widetilde{\mathbf{H}}^{-1}$) is intractable, as it requires storing and inverting the GGN $\widetilde{\mathbf{H}}$. However, we can approximate posterior marginal variances with the diagonal Laplace approximation $\mathrm{diag}(\widetilde{\mathbf{H}}^{-1}) \approx \mathrm{diag}(\widetilde{\mathbf{H}})^{-1}$ (Denker and LeCun, 1990; Kirkpatrick et al., 2017), diagonal Stochastic Weight Averaging Gaussian (SWAG) (Maddox et al., 2019), or even Mean-Field Variational Inference (MFVI) (Blundell et al., 2015; Osawa et al., 2019). In this chapter, we rely on the former two, as the latter involves larger overhead.

It may seem that we have resorted to the poorly performing diagonal assumptions that we sought to avoid in the first place (Ovadia et al., 2019; Foong et al., 2020; Ashukha et al., 2020). However, there is a key difference. We make the diagonal assumption during *subnetwork selection* rather than *inference*; we do full covariance inference over $\mathbf{w}_S$. In Section 4.6, we provide evidence that making a diagonal assumption during subnetwork selection is reasonable by showing that 1) it is substantially less harmful to predictive performance than making the same assumption during inference, and 2) it outperforms random subnetwork selection.

4.6 Experiments

We empirically assess the effectiveness of subnetwork inference compared to methods that do less expressive inference over the full network as well as state-of-the-art methods

for uncertainty quantification in deep learning. We consider three benchmark settings: 1) small-scale toy regression, 2) medium-scale tabular regression, and 3) image classification with ResNet-18. Further experimental results and setup details are presented in Section B.2, respectively.

4.6.1 How does Subnetwork Inference preserve Posterior Predictive Uncertainty?

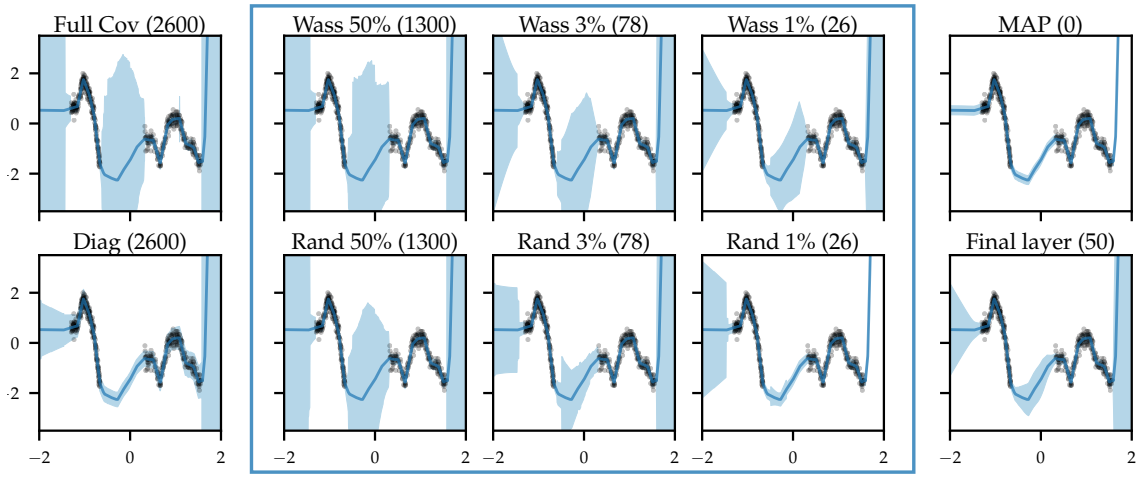


Figure 4.2: Predictive distributions (mean ± std. dev.) for 1D regression. The numbers in parentheses denote the number of parameters over which inference was done (out of 2600 total). The blue box highlights subnetwork inference using Wasserstein (top) and random (bottom) subnetwork selection. Wasserstein subnetwork inference maintains richer predictive uncertainties at smaller parameter counts.

We first assess how the predictive distribution of a full-covariance Gaussian posterior over a selected subnetwork qualitatively compares to that obtained from 1) a full-covariance Gaussian over the *full* network (*Full Cov*), 2) a *factorised* Gaussian posterior over the full network (*Diag*), 3) a full-covariance Gaussian over only the (*Final layer*) of the network (Snoek et al., 2015), and 4) a point estimate (*MAP*). For subnetwork inference, we consider both Wasserstein (*Wass*) (as described in Section 4.5) and uniform random subnetwork selection (*Rand*) to obtain subnetworks that comprise only 50%, 3%, and 1% of the model parameters. For this toy example, it is tractable to compute exact posterior marginal variances to guide subnetwork selection.

Our NN consists of 2 ReLU hidden layers with 50 hidden units each. We employ a homoscedastic Gaussian likelihood function where the noise variance is optimised with maximum likelihood. We use GGN-Laplace inference over network weights (not biases)

in combination with the linearised predictive distribution in (4.22). Thus, all approaches considered share their predictive mean, allowing better comparison of their uncertainty estimates. We set the full network prior precision to $\lambda = 3$ (a value which we find to work well empirically) and set $\lambda_S = \lambda \cdot {}^{S}/_{D}$.

We use a synthetic 1D regression task with two separated clusters of inputs (Antorán et al., 2020), allowing us to probe for 'in-between' uncertainty (Foong et al., 2019). Results are shown in Figure 4.2. Subnetwork inference preserves more of the uncertainty of full network inference than diagonal Gaussian or final layer inference while doing inference over fewer weights. By capturing weight correlations, subnetwork inference retains uncertainty in between clusters of data. This is true for both random and Wasserstein subnetwork selection. However, the latter preserves more uncertainty with smaller subnetworks. Finally, the strong superiority over diagonal Laplace shows that making a diagonal assumption for subnetwork selection but then using a full-covariance Gaussian for inference (as we do) performs significantly better than making a diagonal assumption for the inferred posterior directly (cf. Section 4.5). These results suggest that **expressive inference over a carefully selected subnetwork retains more predictive uncertainty than crude approximations over the full network**.

4.6.2 Subnetwork Inference in Large Models vs Full Inference over Small Models

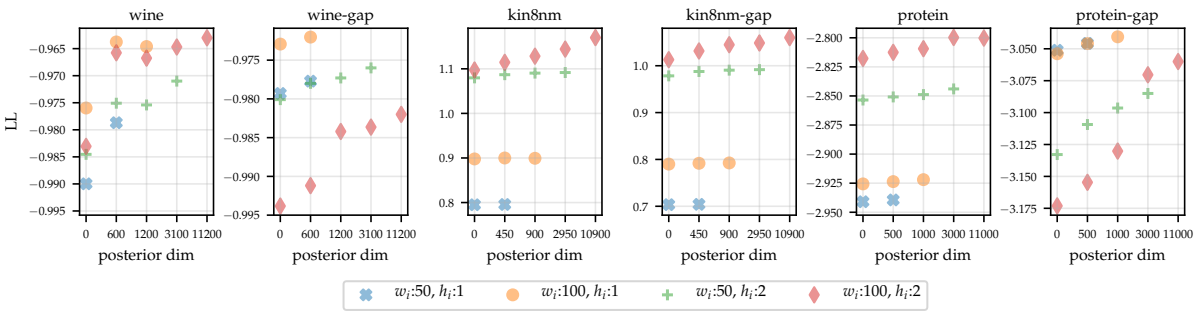


Figure 4.3: Mean test LL values obtained on UCI datasets across all splits. Different markers indicate models with different numbers of weights. The horizontal axis indicates the number of weights over which full covariance inference is performed. 0 corresponds to MAP parameter estimation, and the rightmost setting for each marker corresponds to full network inference.

Secondly, we study how subnetwork inference in larger NNs compares to full network inference in smaller ones. We explore this by considering 4 fully connected NNs of increasing size. These have numbers of hidden layers $h_d = \{1, 2\}$ and hidden layer widths $w_d = \{50, 100\}$. For a dataset with input dimension i_d, the number of weights is given

by $D = (i_d + 1)w_d + (h_d - 1)w_d^2$. Our 2 hidden layer, 100 hidden unit NNs have a weight count of the order 10^4. Full covariance inference in these NNs borders the limit of computational tractability on commercial hardware. We first obtain a MAP estimate of each NN's weights and our homoscedastic likelihood function's noise variance. We then perform full network GGN-Laplace inference for each NN. We also use our proposed Wasserstein rule to prune every NN's weight variances such that the number of variances that remain matches the size of every smaller NN under consideration. We employ the diagonal Laplace approximation to cheaply estimate posterior marginal variances for subnetwork selection. We employ the linearisation in (4.16) and (4.22) to compute predictive distributions. Consequently, NNs with the same number of weights make the same mean predictions. Increasing the number of weight variances considered will thus only increase predictive uncertainty.

We employ 3 tabular datasets of increasing size (input dimensionality, n. points): wine (11, 1439), kin8nm (8, 7373), and protein (9, 41157). We consider their standard train-test splits (Hernández-Lobato and Adams, 2015) and their gap variants (Foong et al., 2019), designed to test for out-of-distribution uncertainty. Details are provided in Section B.2.4. For each split, we set aside 15% of the train data as a validation set. We use these for early stopping when finding MAP estimates and for selecting the weights' prior precision. We keep other hyperparameters fixed across all models and datasets. Results are shown in Figure 4.3.

We present mean test LL values, as these take into account both accuracy and uncertainty. Larger ($w_d = 100, h_d = 2$) models tend to perform best when combined with full network inference, although Wine-gap and Protein-gap are exceptions. Interestingly, these larger models are still best when we perform inference over subnetworks of the size of smaller models. We conjecture this is due to an abundance of degenerate directions (i.e., weights) in the weight posterior NN models (Maddox et al., 2020). Full network inference in small models captures information about both useful and non-useful weights. In larger models, our subnetwork selection strategy allows us to dedicate a larger proportion of our resources to modelling informative weight variances and covariances. In 3 out of 6 datasets, we find abrupt increases in LL as we increase the number of weights over which we perform inference, followed by a plateau. Such plateaus might be explained by most of the informative weight variances having already been accounted for. Considering that the cost of computing the GGN dominates that of NN training, these results suggest that, **given the same amount of compute, it is better to perform subnetwork inference in larger models than full network inference in small ones.**

4.6.3 Image Classification under Distribution Shift

We now assess the robustness of large convolutional neural networks with subnetwork inference to distribution shift on image classification tasks compared to the following baselines: point-estimated networks (MAP), Bayesian deep learning methods that do less expressive inference over the full network: Monte Carlo (MC) Dropout (Gal and Ghahramani, 2016), diagonal Laplace, Variational Online Gauss-Newton (VOGN) (Osawa et al., 2019) (all of which assume factorisation of the weight posterior), and SWAG (Maddox et al., 2019) (which assumes a diagonal plus low-rank posterior). We also benchmark deep ensembles (Lakshminarayanan et al., 2017). The latter is considered state-of-the-art for uncertainty quantification in deep learning (Ovadia et al., 2019; Ashukha et al., 2020). We use ensembles of 5 NNs, as suggested by (Ovadia et al., 2019), and 16 samples for MC Dropout, diagonal Laplace, and SWAG. We use a Dropout probability of 0.1 and a prior precision of $\lambda = 4 \times 10^4$ for diagonal Laplace, found via grid search. We apply all approaches to ResNet-18 (He et al., 2016a), which is composed of an input convolutional block, 8 residual blocks, and a linear layer, for a total of 11,168,000 parameters.

For subnetwork inference, we compute the linearised predictive distribution in (4.23). We use Wasserstein subnetwork selection to retain only 0.38% of the weights, yielding a subnetwork with only 42,438 weights. This is the largest subnetwork for which we can tractably compute a full covariance matrix. Its size is $42,438^2 \times 4$ Bytes ≈ 7.2 GB. We use diagonal SWAG (Maddox et al., 2019) to estimate the marginal weight variances needed for subnetwork selection. We tried diagonal Laplace but found that the selected weights were those where the Jacobian of the NN evaluated at the train points was always zero (i.e., dead ReLUs). The posterior variance of these weights is large, as it matches the prior. However, these weights have little effect on the NN function. SWAG does not suffer from this problem as it disregards weights with zero training gradients. We use a prior precision of $\lambda = 500$, found via grid search.

To assess to importance of principled subnetwork selection, we also consider the baseline where we select the subnetwork uniformly at random (called *Ours (Rand)*). We perform the following two experiments, with results in Figure 4.4.

Rotated MNIST: Following Ovadia et al. (2019); Antorán et al. (2020), we train all methods on MNIST and evaluate their predictive distributions on increasingly rotated digits. While all methods perform well on the original MNIST test set, their accuracy degrades quickly for rotations larger than 30 degrees. In terms of LL, ensembles perform best out of our baselines. Subnetwork inference obtains significantly larger LL values than almost all baselines, including ensembles. The only exception is VOGN, which

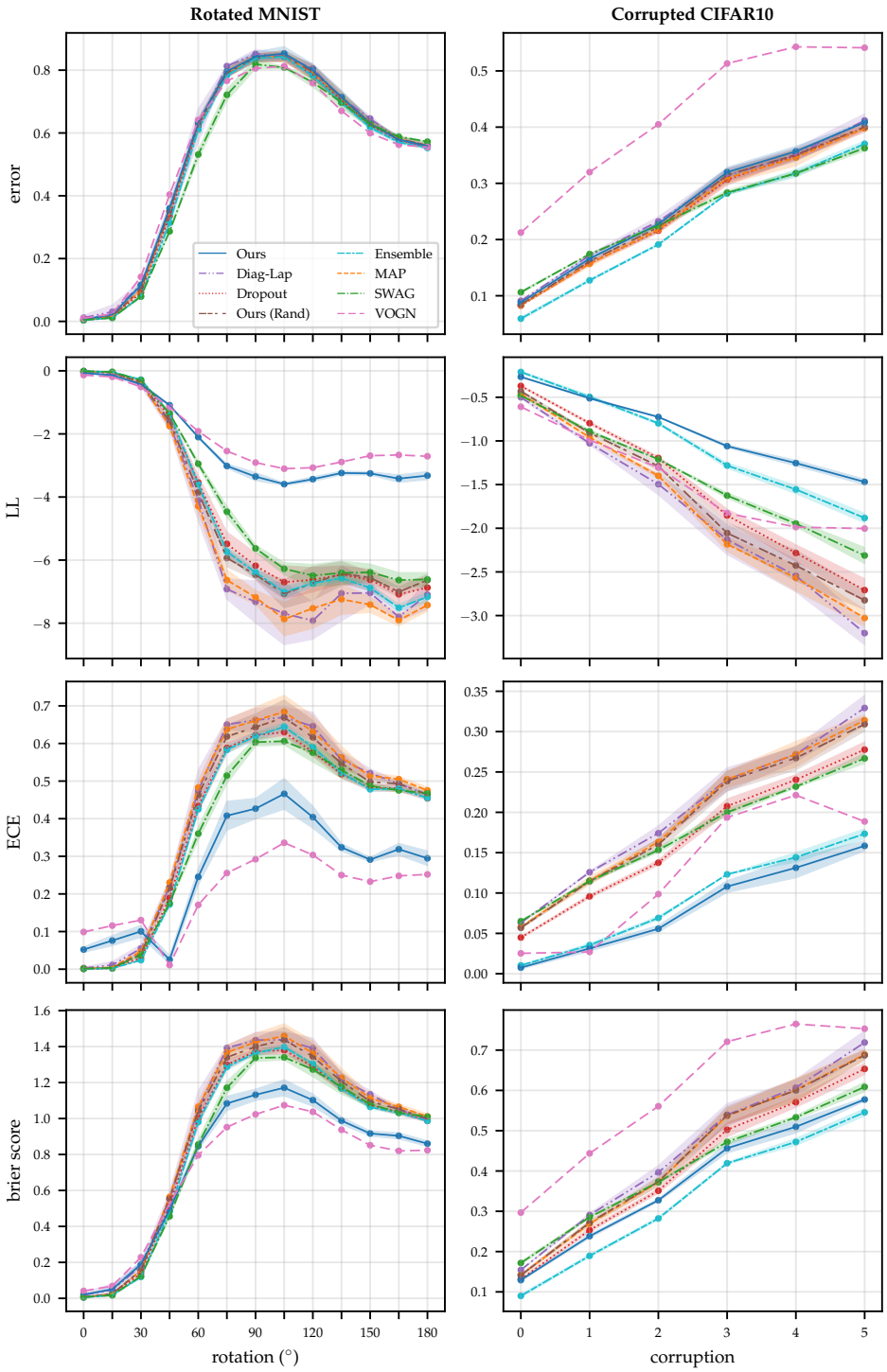


Figure 4.4: Results on the rotated MNIST (left) and the corrupted CIFAR (right) benchmarks, showing the mean ± std. dev. of the error, LL, Expected Calibration Error (ECE), and Brier score (top-to-bottom) across three different seeds. Subnetwork inference retains better uncertainty calibration and robustness to distribution shift than point-estimated networks and other Bayesian deep learning approaches.

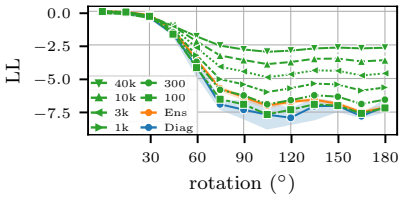
achieves slightly better performance. It was also observed in Ovadia et al. (2019) that MFVI (which VOGN is an instance of) is very strong on MNIST, but its performance deteriorates on larger datasets. Subnetwork inference makes accurate predictions in-distribution while assigning higher uncertainty than the baselines to out-of-distribution points.

Corrupted CIFAR: Again following Ovadia et al. (2019); Antorán et al. (2020), we train on CIFAR10 and evaluate on data subject to 16 different corruptions with 5 levels of intensity each (Hendrycks and Dietterich, 2019). Our approach matches a MAP estimated network in terms of predictive error, as local linearization makes their predictions the same. Ensembles and SWAG are the most accurate. Even so, subnetwork inference differentiates itself by being the least overconfident, outperforming all baselines in terms of LL at all corruption levels. Here, VOGN performs rather badly; while this might appear to contrast with its strong performance on the MNIST benchmark, the behaviour that MFVI performs well on MNIST but poorly on larger datasets was also observed in Ovadia et al. (2019).

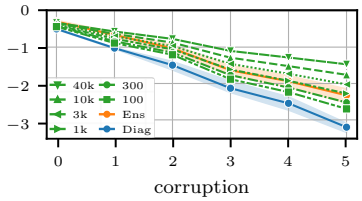
Furthermore, on both benchmarks, we find that randomly selecting the subnetwork performs substantially worse than using our more sophisticated Wasserstein subnetwork selection strategy. This highlights the importance of the way the subnetwork is selected. Overall, these results suggest that **subnetwork inference results in better uncertainty calibration and robustness to distribution shift than other popular uncertainty quantification approaches**.

What about smaller subnetworks? One might wonder if a subnetwork of ∼40K weights is actually necessary. In Figure 4.5, we show that one can also retain strong calibration with significantly smaller subnetworks. Full covariance inference in a ResNet-18 would require storing $\sim 11.2\text{M}^2$ params (∼500TB). Subnet inference reduces the cost (on top of MAP) to as little as 1K^2 params (∼4.0MB) while remaining competitive with deep ensembles. This suggests that subnetwork inference can allow otherwise intractable inference methods to be applied to even larger NNs.

To further validate that we can apply subnetwork inference to larger models, we briefly consider a ResNet-50. We use a subnetwork containing 39,190 / 23,466,560 (0.167%) parameters, which is roughly the same number of parameters as in our ResNet-18 experiments. Figure 4.6 shows our results for this setting. Subnetwork inference in ResNet-50 improves upon a simple MAP estimate of the weights in terms of both LL and calibration metrics. As expected, however, for ResNet-50 the improvement over MAP



(a) Rotated MNIST



(b) Corrupted CIFAR10

Subnet Size	Memory
11.2M (100%)	500**TB**
40K (0.36%)	6.4**GB**
1K (0.01%)	4.0**MB**
100 (0.001%)	40**KB**

(c) Memory Footprints

Figure 4.5: Log-likelihoods of our method with subnetwork sizes between 100-40K using ResNet-18 on rotated MNIST (left) and corrupted CIFAR10 (middle), vs. **Ens**embles and **Diag**onal Laplace, and respective covariance matrix memory footprints (right). For all subnetwork sizes, we use the same hyperparameters as in Section 4.6.3 (i.e., no individual tuning per size). Performance degrades smoothly with subnetwork size, but our method retains strong calibration even with very small subnetworks (requiring only marginal extra memory).

is smaller than for ResNet-18, where we were able to choose a subnetwork containing 0.38% of the parameters.

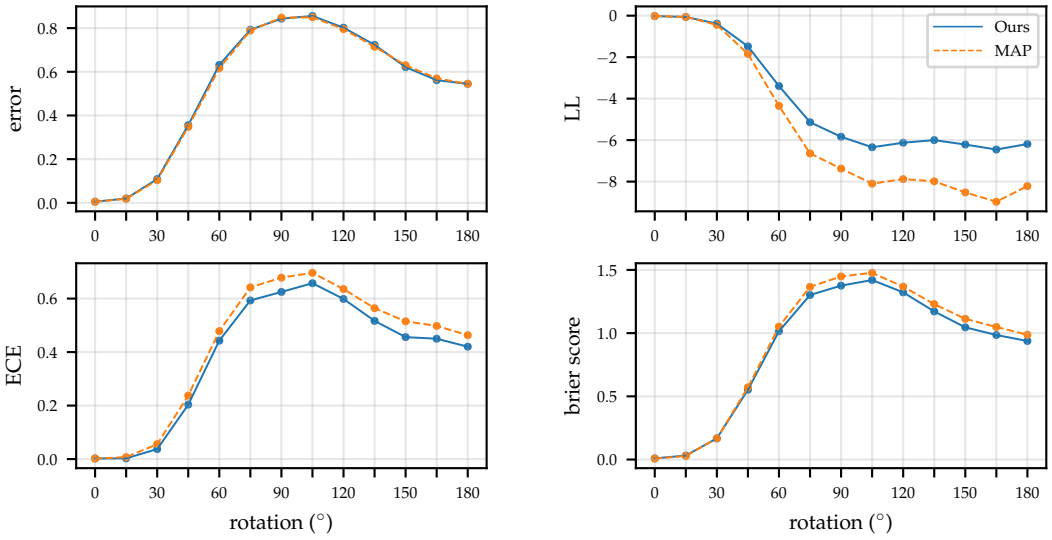


Figure 4.6: MNIST rotation results for ResNet-50, reporting predictive error, LL, ECE, and brier score. We choose a subnetwork containing only 0.167% (39,190 / 23,466,560) of the parameters of the full network.

Comparing the Parameter Efficiency of Subnetwork Linearised Laplace with Deep Ensembles Despite the promising results provided by Subnetwork Linearised Laplace, we note that our method has a notably larger space complexity than our baselines. We therefore investigate the parameter efficiency of our method.

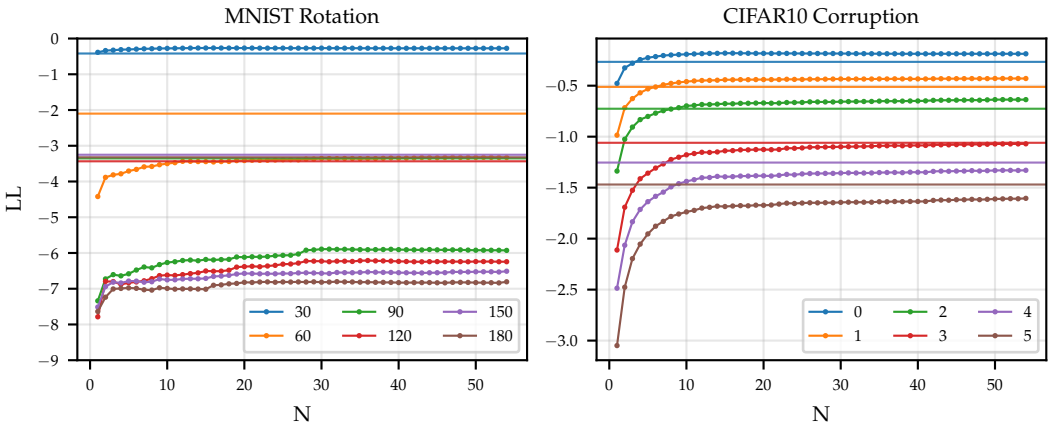


Figure 4.7: Rotated MNIST (left) and Corrupted CIFAR10 (right) results for deep ensembles (Lakshminarayanan et al., 2017) with large numbers of ensemble members (i.e., up to 55). The horizontal axis denotes the number of ensemble members, and the vertical axis denotes performance in terms of log-likelihood. Horizontal lines correspond to the performance of our method, as a reference. Colours denote different levels of rotation (left) and corruption (right).

Our ResNet-18 Model has ∼11.2M parameters. Our subnetwork's covariance matrix contains $42{,}438^2$ parameters. This totals ∼1,830M parameters. This same amount of memory could be used to store around 163 ensemble elements. In Figure 4.7 we compare our subnetwork Linearised Laplace model with increasingly large ensembles on both rotated MNIST and corrupted CIFAR10. Although the performance of ensembles improves as more networks are added, it plateaus around 15 ensemble elements. This is in agreement with the findings of recent works (Antorán et al., 2020; Ashukha et al., 2020; Lobacheva et al., 2020). At large rotations and corruptions, the log likelihood obtained by Subnetwork Linearised Laplace is greater than the asymptotic value obtained by ensembles. This suggests that using a larger number of parameters in an approximate posterior covariance matrix is a more efficient use of space than saving a large number of ensemble elements. We also note that inference in a very large ensemble requires performing a forward pass for every ensemble element. On the other hand, Linearised Laplace requires performing one backward pass for every output dimension and one forward pass.

Out-of-Distribution Rejection In this section we provide additional results on Out-of-distribution (OOD) rejection using predictive uncertainty. First, we train our models on a source dataset. We then evaluate them on the test set from our source dataset and on the test set of a target (OOD) dataset. We expect predictions for the target dataset to be more uncertain than those for the source dataset. Using predictive uncertainty as the discriminative variable, we compute the area under the ROC curve

for each method under consideration and display them in Table 4.1. The CIFAR-SVHN and MNIST-Fashion dataset pairs are chosen following Nalisnick et al. (2019b). On the CIFAR-SVHN task, all methods perform similarly, except for ensembles, which clearly do best. On MNIST-Fashion, SWAG performs best, followed by Subnetwork Linearised Laplace and ensembles.

Table 4.1: AUC-ROC scores for out-of-distribution detection, using CIFAR10 vs SVHN and MNIST vs FashionMNIST as in- (source) and out-of-distribution (target) datasets, respectively.

Source	Target	Ours	Ours (Rand)	Dropout	Diag-Lap	Ensemble	MAP	SWAG
CIFAR10	SVHN	0.85 ±0.03	0.86 ±0.02	0.85 ±0.01	0.86 ±0.02	0.91 ±0.00	0.86 ±0.02	0.83 ±0.00
MNIST	Fashion	0.92 ±0.05	0.75 ±0.02	0.82 ±0.12	0.75 ±0.01	0.90 ±0.09	0.72 ±0.03	0.97 ±0.01

We also simulate a realistic OOD rejection scenario (Filos et al., 2019) by jointly evaluating our models on an in-distribution and an OOD test set. We allow our methods to reject increasing proportions of the data based on predictive entropy before classifying the rest. All predictions on OOD samples are treated as incorrect. Following (Nalisnick et al., 2019b), we use CIFAR10 vs SVHN and MNIST vs FashionMNIST as in- and out-of-distribution datasets, respectively. Note that the SVHN test set is randomly subsampled down to a size of 10,000 to match that of CIFAR10. The results are shown in Figure 4.8. On CIFAR-SVHN all methods perform similarly, with exceptions being ensembles, which perform best, and SWAG, which does worse. On MNIST-FashionMNIST, SWAG performs best, followed by Subnetwork Linearised Laplace. All other methods fail to distinguish very uncertain in-distribution data from low uncertainty OOD points.

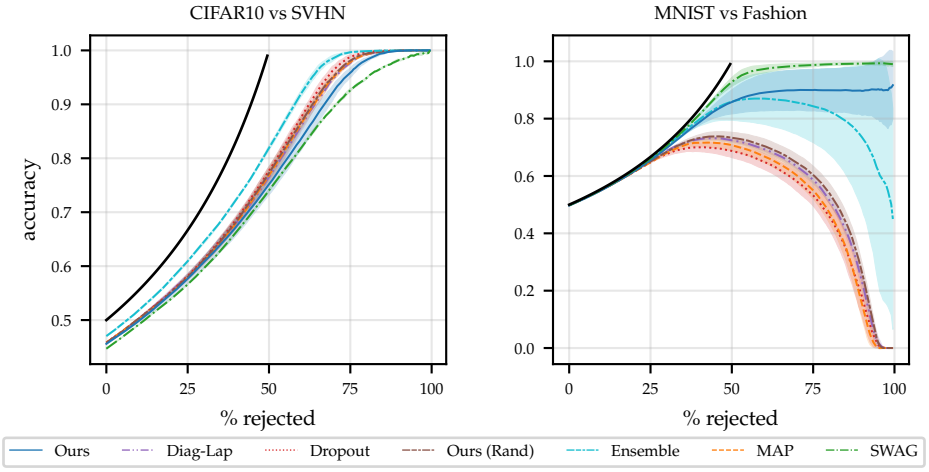


Figure 4.8: Rejection-classification plots.

4.7 Scope and Limitations

Jacobian computation in multi-output models remains challenging. With reverse mode automatic differentiation used in most deep learning frameworks, it requires as many backward passes as there are model outputs. This prevents using linearised Laplace in settings like semantic segmentation (Liu et al., 2019) or classification with large numbers of classes (Deng et al., 2009). Note that this issue applies to the linearised Laplace method and that other inference methods, without this limitation, could be used in our framework.

The choice of prior precision λ determines the performance of the Laplace approximation to a large degree. Our proposed scheme to update λ for subnetworks relies on having a sensible parameter setting for the full network. Since inference in the full network is often intractable, currently the best approach for choosing λ is cross-validation using the subnetwork approximation directly.

The space requirements for the Hessian limit the maximum number of subnetwork weights. For example, storing a Hessian for 40K weights requires around 6.4GB of memory. For very large models, like modern transformers, tractable subnetworks would represent a vanishingly small proportion of the weights. While we demonstrated that strong performance does not necessarily require large subnetworks (see Figure 4.5), finding better subnetwork selection strategies remains a key direction for future research.

4.8 Related Work

Bayesian Deep Learning. There have been significant efforts to characterise the posterior distribution over NN weights $p(\mathbf{w} \mid \mathcal{D})$. Although asymptotically unbiased, sampling-based approaches, such as Hamiltonian Monte Carlo (HMC) (Neal, 1995), are difficult or impractical to scale to large datasets (Betancourt, 2015; Izmailov et al., 2021b). As a result, approaches that find the best surrogate posterior among an approximating family (most often Gaussian) have gained popularity. The first of these was the Laplace approximation, introduced by MacKay (1992), who also proposed approximating the predictive posterior with that of the linearised model (Khan et al., 2019; Immer et al., 2021b); see Daxberger et al. (2021a) for a review of recent advances to use the Laplace approximation in deep learning. The popularisation of larger NN models has made surrogate distributions that capture correlations between weights computationally intractable. Thus, most modern methods make use of the mean field assumption (Blundell et al., 2015; Hernández-Lobato and Adams, 2015; Gal and Ghahramani, 2016; Mishkin et al.,

2018; Osawa et al., 2019). This comes at the cost of limited expressivity (Foong et al., 2020) and empirical under-performance (Ovadia et al., 2019; Antorán et al., 2020). We note that Farquhar et al. (2020) argue that in deeper networks the mean-field assumption should not be restrictive. Our empirical results seem to contradict this proposition. We find that scaling up approximations that *do* consider weight correlations (e.g., MacKay (1992); Louizos and Welling (2016); Ritter et al. (2018); Maddox et al. (2019)) by lowering the dimensionality of the weight space outperforms diagonal approximations. We conclude that more research is warranted in this area. Finally, recent works have demonstrated the benefit of capturing the multi-modality of the posterior distribution via ensembles/mixtures (Lakshminarayanan et al., 2017; Fort et al., 2019; Filos et al., 2019; Wilson and Izmailov, 2020; Eschenhagen et al., 2021).

Neural Linear Methods. These represent a generalised linear model in which the basis functions are defined by the $l-1$ first layers of a NN. That is, neural linear methods perform inference over only the last layer of a NN, while keeping all other layers fixed (Snoek et al., 2015; Riquelme et al., 2018; Ovadia et al., 2019; Ober and Rasmussen, 2019; Pinsler et al., 2019; Kristiadi et al., 2020). They can also be viewed as a special case of subnetwork inference, in which the subnetwork is simply defined to be the last NN layer.

Inference over Subspaces. The subfield of NN pruning aims to increase the computational efficiency of NNs by identifying the smallest subset of weights that are required to make accurate predictions; see e.g., Frankle and Carbin (2019); Wang et al. (2020). Our work differs in that it retains all NN weights but aims to find a small subset over which to perform probabilistic reasoning. More closely related work to ours is that of Izmailov et al. (2019), who propose to perform inference over a low-dimensional subspace of weights; e.g., one constructed from the principal components of the Stochastic Gradient Descent (SGD) trajectory. Moreover, several recent approaches use low-rank parameterisations of approximate posteriors in the context of variational inference (Rossi et al., 2020; Swiatkowski et al., 2020; Dusenberry et al., 2020a). This could also be viewed as doing inference over an implicit subspace of weight space. In contrast, we propose a technique to find subsets of weights which are relevant to predictive uncertainty, i.e., we identify axis-aligned subspaces.

4.9 Summary

This chapter has three main findings (1) modelling weight correlations in NNs is crucial to obtaining reliable predictive posteriors, (2) given these correlations, unimodal approximations of the posterior can be competitive with approximations that assign mass to

multiple modes (e.g., deep ensembles), (3) inference does not need to be performed over all the weights in order to obtain reliable predictive posteriors.

We use these insights to develop a framework for scaling Bayesian inference to NNs with many weights. We approximate the posterior over a subset of the weights while keeping all others deterministic. Computational cost is decoupled from the total number of weights, allowing us to conveniently trade it off with the quality of approximation. This allows us to use more expressive posterior approximations, such as full-covariance Gaussian distributions.

Linearised Laplace subnetwork inference can be applied post-hoc to any pre-trained model, making it particularly attractive for practical use. Our empirical analysis suggests that this method (1) is more expressive and retains more uncertainty than crude approximations over the full network, (2) allows us to employ larger NNs, which fit a broader range of functions, without sacrificing the quality of our uncertainty estimates, and (3) is competitive with state-of-the-art uncertainty quantification methods, like deep ensembles.

In this thesis, we have now introduced two methods for improving uncertainty estimation for NNs, namely Depth Uncertainty Networks (DUNs) and the Subnetwork Linearised Laplace approximation. But, we did not compare these two methods directly. However, using deep ensembles as a common baseline in the rotated MNIST and corrupted CIFAR benchmarks allows us to indirectly compare these methods. While both DUNs and subnetwork inference outperform ensembles on rotated MNIST, subnetwork inference provides a much larger performance boost. On the other hand, for corrupted CIFAR, DUNs performs worse than ensembles, while subnetwork inference performs slightly better. These results indicate that subnetwork inference provides better uncertainty estimates, especially as the dataset becomes more complicated. However, there are still settings in which DUNs might be a more suitable method. For instance, where the neural network is sufficiently overparameterised relative to the dataset, or when the relative simplicity of implementing DUNs and/or the lower memory footprint of the method are important factors.

While subnetwork inference is a promising approach to scaling Bayesian inference to large NNs, we saw in Figure 4.5 and Figure 4.6 that as the fraction of weights for which we perform inference decreases, the performance of our method degrades. In the next chapter, we will explore uncertainty quantification for NNs with *very* large numbers of weights. We will do this by constructing efficient ensembles that rely on marginalisation over different explanations of the data rather than inference over any parameters.

Chapter 5

Sparse-MoEs meet Efficient Ensembles

In this chapter, we continue to consider uncertainty estimation for NNs. However, the setting we will consider is that of extremely large sparse Mixture of Expert (MoE) models (with up to 2.7 *billion* parameters). As a result, even the "scalable" Bayesian inference method developed in the previous chapter would only allow us to perform inference over a vanishingly small fraction of the parameters. Instead, we take the perspective of Wilson (2020); Wilson and Izmailov (2020) that marginalisation is perhaps more important than Bayesian inference for strong uncertainty estimation performance. To this end, we study the interplay between sparse MoEs—which adaptively combine multiple "expert" subnetworks' explanations of the data when making predictions—and ensembles—which combine independently trained models' predictions. This results in two sets of contributions:

Contribution 1: Complementarity of sparse MoEs and ensembles. In Section 5.3 we compare and contrast sparse MoEs and ensembles. Later, in Section 5.5, we show that sparse MoEs and ensembles have complementary features and benefit from each other. Specifically:

- The adaptive computation in sparse MoEs and the static combination in ensembles are orthogonal, with additive benefits when used together. At the intersection of these two model families is an exciting trade-off between performance and compute (i.e., as measured in Floating Point Operations Per Second (FLOPs)). That is, the frontier can be mapped out by varying the ensemble size and the sparsity of MoEs.

- Over tasks where either sparse MoEs or ensembles are known to perform well, naive—and computationally expensive—ensembles of MoEs provide the best predictive performance. Our benchmarking effort includes the first evaluation of sparse MoEs

on uncertainty-related vision tasks, which builds upon the work of Riquelme et al. (2021).

Contribution 2: Efficient ensemble of experts. In Section 5.4, we propose Efficient Ensemble of Experts (Efficient Ensemble of Experts (E^3)), see Figure 5.1, an efficient ensemble approach tailored to sparse MoEs. In Section 5.5, we provide evidence that

- E^3 improves over sparse MoEs across few-shot error, likelihood and calibration error. E^3 matches the performance of deep ensembles while using from 30% to 45% fewer FLOPs.
- E^3 gracefully scales up to 2.7B parameter models.
- E^3 is both simple—requiring only minor implementation changes—and convenient—E^3 models can be fine-tuned directly from standard sparse-MoE checkpoints.

This chapter is based on the paper "Sparse MoEs meet Efficient Ensembles" (Allingham et al., 2022b), which was written in collaboration with Florian Wenzel, Zelda Mariet, Basil Mustafa, Joan Puigcerver, Neil Houlsby, Vincent Fortuin, Balaji Lakshminarayanan, Jasper Snoek, Dustin Tran, Carlos Riquelme Ruiz, and Rodolphe Jenatton. Rodolphe, Carlos, and Dustin, initiated the project and all provided supervision. Rodolphe and I were both heavily involved with conceptualisation, exploration, coding, evaluation, and presentation of the results, as well as writing the paper.

5.1 Motivation

NNs typically use all of their parameters to process an input. Sustaining the growth of such models—reaching today up to 100B+ parameters (Brown et al., 2020)—is challenging, e.g., due to their high computational and environmental costs (Strubell et al., 2019; Patterson et al., 2021). In this context, sparse MoEs employ *conditional computation* (Bengio et al., 2013) to combine multiple submodels and route examples to specific "expert" submodels (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022; Riquelme et al., 2021). Conditional computation can decouple the growth of the number of parameters from the training and inference costs, by only activating a subset of the overall model in an input-dependent fashion.

Paralleling this trend, the deployment of machine learning systems in safety-critical fields, e.g., medical diagnosis (Dusenberry et al., 2020b) and self-driving cars (Levinson et al., 2011), has motivated the development of reliable deep learning, e.g., for *calibrated*

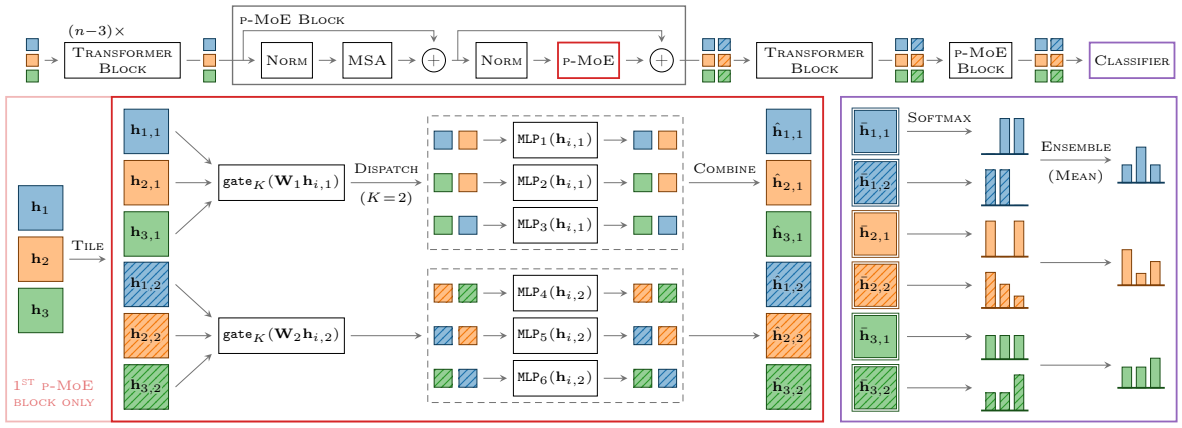


Figure 5.1: End-to-end overview of E^3 with $E = 6$ experts, partitioned into $M = 2$ groups, with sparsity of $K = 2$, and a "last-2" configuration. **Top**: E^3 contains a sequence of transformer blocks, followed by alternating transformer and p(artitioned)-MoE blocks. As in ViT, images are split into patches whose embeddings are processed by each block. Here, we show 1 embedding for each of three images (■, ■, ■). **Bottom left**: In a p-MoE block, we replace the transformer block's MLP with parallel partitioned expert MLP, see (5.2). The effect of the routing weights is not depicted. Embeddings are tiled (▨) in the first p-MoE block only. **Bottom right**: The classifier averages predictions from the final tiled representations (▣).

and robust predictions (Ovadia et al., 2019). Among the existing approaches, ensembles of NNs have remarkable performance for calibration and accuracy under dataset shifts (Ovadia et al., 2019). These methods improve reliability by aggregating the predictions of individual submodels, referred to as ensemble members. However, this improvement comes at a significant computational cost. Hence, naively ensembling NNs that continue to grow in size becomes less and less feasible. In this chapter, we try to overcome this limitation. Our core motivation is to improve the robustness and uncertainty estimates of large-scale fine-tuned models through ensembling, but to do so in a tractable—and thus practically useful—manner, by carefully developing a hybrid approach using advances in sparse MoEs.

While sharing conceptual similarities, these two classes of models—MoEs and ensembles—have different properties. Sparse MoEs adaptively combine their experts depending on the inputs, and the combination generally happens at internal activation levels. Ensembles typically combine several models in a static way and at the prediction level. Moreover, these two classes of models tend to be benchmarked on different tasks: few-shot classification for MoEs (Riquelme et al., 2021) and uncertainty-related evaluation for ensembles (Ovadia et al., 2019; Gustafsson et al., 2020). For example, sparse MoEs are seldom, if ever, applied to the problems of calibration. Thus, we are also motivated to better understand the interplay between these two classes of models.

5.2 Preliminaries

We focus on classification tasks where we learn classifiers of the form $f(\boldsymbol{x}; \boldsymbol{\theta})$ based on some training data $\mathcal{D} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$. A pair $(\boldsymbol{x}_n, y_n)$ corresponds to an input $\boldsymbol{x}_n \in \mathbb{R}^P$ together with its label $y_n \in \{1, \dots, C\}$ belonging to one of the C classes. The model $f(\cdot; \boldsymbol{\theta})$ is parameterised by $\boldsymbol{\theta}$ and outputs a C-dimensional probability vector. We use $\circ$ to refer to the matrix element-wise product.

5.2.1 Vision Transformers and Sparse MoEs

Vision Transformers. Throughout this chapter, we choose the model f to be a Vision Transformer (ViT) (Dosovitskiy et al., 2021). ViT is growing in popularity for vision, especially in transfer-learning settings, where it was shown to outperform convolutional networks while requiring fewer pre-training resources. ViT operates at the level of patches. An input image is split into equal-sized patches (e.g., 32×32, 16×16, or 14×14 pixels) whose resulting sequence is (linearly) embedded and processed by a Transformer (Vaswani et al., 2017). The operations in the Transformer then mostly consist of a succession of Multi-headed Self-Attention (MSA) and Multi-Layer Perceptron (MLP) layers. ViT is defined at different scales (Dosovitskiy et al., 2021): S(mall), B(ase), L(arge) and H(uge); see specifications in Section C.1.1. For example, ViT-L/16 stands for a large ViT with patch size 16×16.

Sparse MoEs and V-MoEs. The main feature of sparsely-gated mixture-of-experts models (sparse MoE) lies in the joint use of sparsity and *conditional computation* (Bengio et al., 2013). In those models, we only activate a small subset of the network parameters *for a given input*, which allows the total number of parameters $\boldsymbol{\theta}$ to grow while keeping the overall computational cost constant. The *experts* are the subparts of the network activated on a per-input fashion.

Central to our study, Riquelme et al. (2021) recently extended ViT to sparse MoEs. Their extension, referred to as Vision-MoE (V-MoE), follows the successful applications of sparse models in NLP (Shazeer et al., 2017). Riquelme et al. (2021) show that V-MoEs dominate their "dense" ViT counterparts on a variety of tasks for the same computational cost. In the specific case of V-MoEs, the experts are placed in the MLP layers of the Transformer, a design choice reminiscent of Lepikhin et al. (2021) in NLP. Given the

input $\mathbf{h} \in \mathbb{R}^D$ of such a layer, the output of a single $\texttt{MLP}(\mathbf{h})$ is replaced by

$$\texttt{MoE}(\mathbf{h}) = \sum_{e=1}^{E} g_e(\mathbf{h}) \cdot \texttt{MLP}_e(\mathbf{h}) \quad \text{with} \quad \{g_e(\mathbf{h})\}_{e=1}^{E} = \texttt{top}_K(\texttt{softmax}(\mathbf{W}\mathbf{h})), \tag{5.1}$$

where the *routing* weights $\{g_e(\mathbf{h})\}_{e=1}^{E}$ combine the outputs of the E different experts $\{\texttt{MLP}_e\}_{e=1}^{E}$. To sparsely select the experts, $\texttt{top}_K$ sets all but the K largest weights to zero. The router parameters $\mathbf{W} \in \mathbb{R}^{E \times D}$ are trained together with the rest of the network parameters. We call the layer defined by (5.1) a MoE layer. In practice, the weights $\{g_e(\mathbf{h})\}_{e=1}^{E}$ are obtained by a noisy version of the routing function $\texttt{top}_K(\texttt{softmax}(\mathbf{W}\mathbf{h} + \sigma \cdot \boldsymbol{\varepsilon}))$ with $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{\epsilon} \,|\, \mathbf{0}, \boldsymbol{I})$, which mitigates the non-differentiability of $\texttt{top}_K$ when combined with auxiliary losses (see Appendix A in Shazeer et al. (2017)). Making non-differentiable operators smooth with some noise injection is an active area of research (Duchi et al., 2012; Abernethy et al., 2016; Berthet et al., 2020). We use the shorthand $\texttt{gate}_K(\mathbf{z}) = \texttt{top}_K(\texttt{softmax}(\mathbf{z} + \sigma \cdot \boldsymbol{\varepsilon}))$ and take $\sigma = 1/E$ as in Riquelme et al. (2021).

In this chapter, we consider the "last-n" setting of Riquelme et al. (2021) wherein only a few MoE layers are placed at the end of the Transformer ($n = 2$ for the {S, B, L} scale and $n = 5$ for H). This setting retains most of the performance gains of V-MoEs while greatly reducing the training cost.

5.2.2 Ensembles of Neural Networks

Ensembles. We build on the idea of ensembles, which is a known scheme to improve the performance of individual models (Hansen and Salamon, 1990; Geman et al., 1992; Krogh and Vedelsby, 1995; Opitz and Maclin, 1999; Dietterich, 2000; Lakshminarayanan et al., 2017). Formally, we assume a set of M model parameters $\Theta = \{\boldsymbol{\theta}_m\}_{m=1}^{M}$. We refer to M as the *ensemble size*. Prediction proceeds by computing $\frac{1}{M}\sum_{\boldsymbol{\theta}\in\Theta} f(\boldsymbol{x}; \boldsymbol{\theta})$, i.e., the average probability vector over the M models. To assess the diversity of the predictions in the ensemble, we will use the KL divergence $\mathbb{D}_{\mathrm{KL}}\left[f(\boldsymbol{x}_t; \boldsymbol{\theta}_m) \,||\, f(\boldsymbol{x}_t; \boldsymbol{\theta}_{m'})\right]$ between the predictive distributions $f(\boldsymbol{x}_t; \boldsymbol{\theta}_m)$ and $f(\boldsymbol{x}_t; \boldsymbol{\theta}_{m'})$, averaged over the test input $\boldsymbol{x}_t$ and all pairs (m, m') of ensemble members.

Batch ensembles. Wen et al. (2020) construct a Batch Ensemble (BE) as a collection of submodels, with the parameters $\boldsymbol{\theta}_m \in \Theta$ sharing components. This mitigates the computational and memory cost of ensembling, while still improving performance. We focus on the example of a single dense layer in f with parameters $\mathbf{U} \in \mathbb{R}^{D \times L}$, assuming no bias. BE defines M copies of parameters $\{\mathbf{U}_m\}_{m=1}^{M}$ so that $\mathbf{U}_m = \mathbf{U} \cdot (\boldsymbol{r}_m \boldsymbol{s}_m^\top)$, where

$\mathbf{U}$ are parameters shared across ensemble members, and $\boldsymbol{r}_m$ and $\boldsymbol{s}_m$ are separate D- and L-dimensional vectors for ensemble member m. Given an input, BE produces M outputs, which are averaged after applying all layers. Despite the simple rank-1 parametrisation, BE leads to remarkable predictive performance and robustness (Wen et al., 2020). Notably, the efficiency of BE relies on both the parameter sharing and the tiling of the inputs to predict with the M ensemble members, two insights that we exploit in this chapter.

5.2.3 Pre-training and Fine-tuning

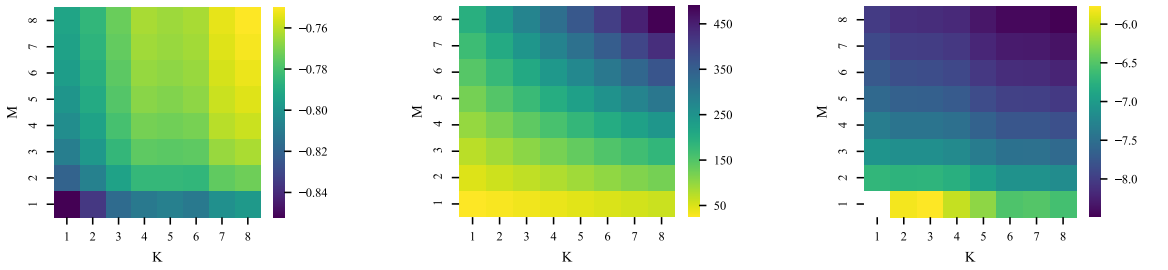
Large-scale Transformers pre-trained on *upstream* tasks were shown to have strong performance when fine-tuned on smaller *downstream* tasks, across a variety of domains (Devlin et al., 2019; Dosovitskiy et al., 2021; Radford et al., 2021). We follow this paradigm and focus on the fine-tuning of models pre-trained on JFT-300M (Sun et al., 2017), similar to Riquelme et al. (2021). We will thus assume the availability of already pre-trained ViT and V-MoE model checkpoints. Our assumption relies on the growing popularity of transfer learning, e.g., Kolesnikov et al. (2020), and the increasing accessibility of pre-trained models in repositories such as `www.tensorflow.org/hub` or `www.pytorch.org/hub`. The fine-tuning of all the approaches we study here, including extensions of ViT and V-MoE, will be either directly compatible with those checkpoints or require only mild adjustments, e.g., reshaping or introducing new downstream-specific parameters (see Section C.2). Also, unless otherwise mentioned, the performance we report will always be downstream, e.g., for ImageNet (Deng et al., 2009) or CIFAR10/100 (Krizhevsky, 2009). In all our comparisons, we will use the downstream training FLOPs, or GFLOPs (i.e., 10^9 FLOPs), to quantify the computational cost of the different methods.

5.3 Sparse MoEs meet Ensembles

As illustrated in Table 5.1, sparse MoEs and ensembles have different properties. For instance, ensembles typically do not use conditional computation and just statically combine members at the prediction level. This contrasts with sparse MoEs where the different experts are combined at internal activation levels while enjoying per-input adaptivity through the routing logic, see (5.1). In terms of cost, sparse MoEs are usually designed to match the inference time of their dense counterparts whereas ensembles, in their simplest forms, will typically lead to a substantial overhead. In this section, we study the extent to which these properties are complementary and may benefit from each other.

Table 5.1: Overview of key properties of sparse MoEs, ensembles, and E^3. E^3 achieves the best of both worlds. `dense` is a base model upon which we add the sparse MoE or ensemble logic, e.g., a ViT model in this chapter.

	Predictions	Combination Level	Conditional Computation	Cost
Sparse MoEs	Single	Activation	Yes, adaptively per-input	$\approx$ `dense`
Ensembles	Multiple	Prediction	No, static	$>$ `dense`
E^3	Multiple	Activation & prediction	Yes, adaptively per-input	$\approx$ `dense`



(a) Log-Likelihood (LL) **(b)** (downstream) GFLOPs **(c)** $\log\left[\frac{\text{LL}_{(K,M)}-\text{LL}_{(1,1)}}{\text{GFLOPs}_{(K,M)}-\text{GFLOPs}_{(1,1)}}\right]$

Figure 5.2: Increasing static (M) and adaptive (K) ensembling on ImageNet, for V-MoE-S/32. Yellow/**purple** indicates better/worse performance. Increasing *both* static and adaptive ensembling is beneficial, the latter being more efficient.

In Section 5.5, we further evaluate this complementarity on tasks where either sparse MoEs or ensembles are known to perform well, e.g., few-shot and Out-of-distribution (OOD) evaluations, respectively.

To investigate the interactions of the properties in Table 5.1, we study the performance of *downstream* deep ensembles (i.e., with all ensemble members having the same upstream checkpoint) formed by M independent V-MoEs with E experts per MoE layer and a sparsity level K (the larger K, the more selected experts). M controls the static combination, while K and E impact the adaptive combination of experts in each sparse MoE model. We report in Figure 5.2 the ImageNet performance and compute cost for ensembles with varying choices of K and M, while keeping $E = 32$ fixed. We focus on K rather than E to explore adaptive computation, as we found the performance quickly plateaus with E (see Figure C.2 in the Appendix). Also, by fixing $E = 32$, we match more closely the setup of Riquelme et al. (2021). The architecture of the V-MoE is ViT-S/32, see details in Section C.1.7. We make the following observations:

Investigating the cumulative effects of adaptive and static ensembling. In the absence of ensembles (i.e., when we consider $M = 1$), and given a fixed number of experts, the authors of Riquelme et al. (2021) already reported an increase in performance as K gets larger. Interestingly, we observe that for each value of K, it is also beneficial to increase the ensemble size M. In other words, the static combination of ensembles is beneficial when applied to sparse MoEs. This observation is perhaps surprising since adaptive combination may already encapsulate the effect of static combination. Figure 5.3, and Section C.8.1, show that the combination of static ensembling and adaptivity is beneficial to Negative Log Likelihood (NLL) for a range of ViT families. We also see that the benefits of static ensembling are similar for V-MoE and ViT (which does not have any adaptivity).

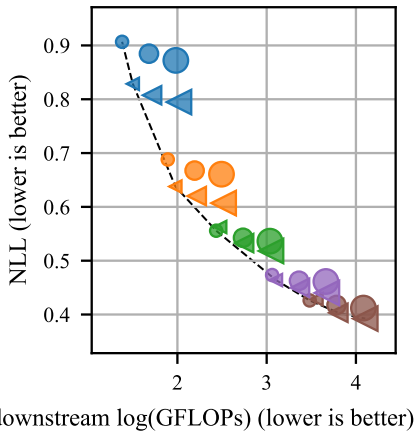


Figure 5.3: ImageNet evaluation for ViT (●) and V-MoE (K=1) (◂) ensembles of size {1, 2, 4} (⬥, ◆, ◆). Model sizes: S/32 (◆), B/32 (◆), L/32 (◆), L/16 (◆), and H/14 (◆). ViT and V-MoE benefit from ensembling equally, at all scales.

Investigating ensembles of sparse MoEs with fewer experts. In Section C.8.2, we compare the performance of a V-MoE with $E = 32$ experts and ensembles of V-MoEs with fewer experts, namely ($M = 2$, $E = 16$) and ($M = 4$, $E = 8$). We see that the performance—e.g., as measured by NLL—is better for ($M = 4$, $E = 8$) than ($M = 2$, $E = 16$) which is in turn better than ($M = 1$, $E = 32$). Thus, we conclude that **reducing the number of experts only mildly affects the combination of adaptive and static ensembling.**

Investigating the trade-off between FLOPs and performance. Without any computational constraints, the previous observation would favour approaches with the largest values of K and M. However, different values of (K, M) lead to different computational costs, as measured here by FLOPs, with $(K, M) = (1, 1)$ being the cheapest. Figure 5.2b shows, as expected, that the number of FLOPs grows more quickly along the M axis than along the K axis. To capture the various trade-offs at play, in Figure 5.2c we report the logarithm of the normalised gains in log likelihood $\frac{\text{LL}_{(K,M)} - \text{LL}_{(1,1)}}{\text{GFLOPs}_{(K,M)} - \text{GFLOPs}_{(1,1)}}$ when going from $(K, M) = (1, 1)$ to other choices of (K, M). Interestingly, it appears more advantageous to first grow K, i.e., the adaptive combination, before growing M.

Summary. While simple ensembling of sparse MoEs results in strong predictive performance, we lose their computational efficiency. We next show how to efficiently combine ensembling and sparse MoEs, exploiting the fact that statically combining sparse MoEs with fewer experts remains effective.

5.4 Efficient Ensemble of Experts

Equipped with the insights from Section 5.3, we describe efficient ensemble of experts (E^3), with the goal of keeping the strengths of both sparse MoEs and ensembles. Conceptually, E^3 jointly learns an ensemble of smaller sparse MoEs, where all layers without experts (e.g., attention layers) are shared across the members.

5.4.1 The Architecture

There are two main components in E^3:

Disjoint subsets of experts. We change the structure of (5.1) by *partitioning* the set of E experts into M subsets of E/M experts (assuming that E is a multiple of M). We denote the subsets by $\mathcal{E}_m$. For example, $\mathcal{E}_1 = \{1, 2, 3\}$ and $\mathcal{E}_2 = \{4, 5, 6\}$ for $E = 6$ and $M = 2$. Intuitively, the ensemble members have separate parameters for independent predictions, while efficiently sharing parameters among all non-expert layers. Instead of having a single routing function $\texttt{gate}_K(\mathbf{W}\cdot)$ as in (5.1), we apply separate routing functions $\texttt{gate}_K(\mathbf{W}_m\cdot)$ to each subset $\mathcal{E}_m$. Note that this does not affect the total number of parameters since $\mathbf{W}$ has E rows while each $\mathbf{W}_m$ has E/M rows. A similar partitioning of the experts was done in Yang et al. (2021) but not exploited to create different ensemble members, in particular not in conjunction with tiled representations, which we show to be required to get performance gains (see Section 5.4.2).

Tiled representation. To jointly handle the predictions of the M ensemble members, we tile the inputs by a factor M, inspired by Wen et al. (2020). This enables a simple implementation of E^3 on top of an existing MoE. In Section C.5, we connect sparse MoEs and BE, illustrating that tiling naturally fits into the formalism of sparse MoEs. Because of the tiling, a given image patch has M different representations that, when entering an MoE layer, are each routed to their respective expert subsets $\mathcal{E}_m$. Formally, consider some tiled inputs $\mathbf{H} \in \mathbb{R}^{B\times M\times D}$ where B refers to the batch size (a batch contains image patches) and $\mathbf{h}_{i,m} \in \mathbb{R}^D$ is the representation of the i-th input for the m-th member. The

routing logic in E^3 can be written as

$$\texttt{p-MoE}(\mathbf{h}_{i,m}) = \sum_{e \in \mathcal{E}_m} g_e(\mathbf{h}_{i,m}) \cdot \texttt{MLP}_e(\mathbf{h}_{i,m}) \quad \text{with} \quad \{g_e(\mathbf{h}_{i,m})\}_{e \in \mathcal{E}_m} = \texttt{gate}_K(\mathbf{W}_m \mathbf{h}_{i,m}), \quad (5.2)$$

where the routing weights are now $\mathbf{W}_m \in \mathbb{R}^{(E/M) \times D}$; see Figure 5.1.

To echo the observations from Section 5.3, we can first see that E^3 brings together the static and adaptive combination of ensembles and sparse MoEs, which we found to be complementary. However, we have seen that static ensembling comes at the cost of a large increase in FLOPs, thus we opt for an efficient ensembling approach. Second, we "split" the MoE layers along the axis of the experts, i.e., from E experts to M times E/M experts. We do so since we observed that the performance of sparse MoEs tends to plateau quickly for more experts. We note that E^3 retains the property of ensembles to output multiple predictions per input.

In a generic implementation, we tile a batch of B inputs $\boldsymbol{X} \in \mathbb{R}^{B \times P}$ by a factor M to obtain the tiled inputs $\boldsymbol{X}_{\text{tiled}} = [\boldsymbol{X}; \ldots; \boldsymbol{X}] \in \mathbb{R}^{(M \cdot B) \times P}$ and the model processes $f(\boldsymbol{X}_{\text{tiled}}; \boldsymbol{\theta})$. Since tiling in E^3 has an effect only from the first MoE layer onwards, we postpone the tiling operation to that stage, thus saving otherwise redundant prior computations in non-MoE-layers. For example, for L/16 and $K = M = 2$, we can save about 47% of the FLOPs. Further implementation details of E^3, and a discussion of the increased memory consumption due to tiling, are in Section C.3. Code can be found at `https://github.com/google-research/vmoe`. Finally, we note that although E^3 and BE share conceptual design similarities—tiled representation and sharing of parameters—they differ in fundamental structural ways, see Section C.6.

5.4.2 Ablation Studies: Partitioning and Tiling

Our method introduces two changes to V-MoEs: (a) the partitioning of the experts and (b) the tiling of the representations. In this section, we assess the separate impact of each of those changes and show that it is indeed their combination that explains the performance gains. We summarise the results of this study in Table 5.2, which shows ImageNet performance—NLL, classification error, Expected Calibration Error (ECE) (Guo et al., 2017), and Kullback-Leibler Divergence (KLD)—for different ablations of E^3. See Section C.10, for end-to-end overview diagrams in the style of Figure 5.1, for these ablations. We use $E = 32$ experts. We provide FLOPs measurements for these ablations in Section C.9.

Table 5.2: ImageNet performance (mean ± std. err. over 8 seeds) of E^3-B/32 ($K = M = 2$), V-MoE ($K = 4$), and two ablations: *only* tiling and *only* partitioning. The noise in $\texttt{gate}_K$ is denoted by σ.

	NLL ↓	Error ↓	ECE ↓	KL ↑
V-MoE	0.636 ±0.001	16.70 ±0.04	0.034 ±0.001	—
E^3	**0.612** ±0.001	**16.49** ±0.02	**0.013** ±0.000	**0.198** ±0.003
Tiling	0.637 ±0.002	16.74 ±0.06	0.028 ±0.001	0.000 ±0.000
Tiling ($\sigma \times 2$)	0.638 ±0.001	16.72 ±0.03	0.033 ±0.001	0.001 ±0.000
Tiling ($\sigma \times 4$)	0.638 ±0.001	16.74 ±0.03	0.033 ±0.001	0.002 ±0.000
Partitioning	0.640 ±0.001	16.72 ±0.05	0.034 ±0.001	—

Partitioning without Tiling

We first compare E^3 with a variant of V-MoE where we only partition the set of experts (*Partitioning*). In this variant, each input $\mathbf{h}_i \in \mathbb{R}^D$ (note the dropping of the index m due to the absence of tiling) can select K experts in subset $\mathcal{E}_m$, resulting in a total of $K \times M$ selected experts per input. Formally, (5.2) becomes

$$\texttt{part-only-MoE}(\mathbf{h}_i) = \sum_{m=1}^{M} \sum_{e \in \mathcal{E}_m} g_e(\mathbf{h}_i) \cdot \texttt{MLP}_e(\mathbf{h}_i) \quad \text{with} \quad \{g_e(\mathbf{h}_i)\}_{e \in \mathcal{E}_m} = \texttt{gate}_K(\mathbf{W}_m \mathbf{h}_i).$$

The *expert prototyping* of Yang et al. (2021) leads to a similar formulation. As shown in Table 5.2, across all metrics, *Partitioning* is not competitive with E^3. We do not report KL since, without tiling, *Partitioning* does not output multiple predictions per input.

Tiling without Partitioning

We now compare E^3 with the variant where only the tiling is enabled (*Tiling*). In this case, we have tiled inputs $\mathbf{H} \in \mathbb{R}^{B \times M \times D}$ applied to the standard formulation of (5.1). Compared with (5.2), there is no mechanism to enforce the M representations of the i-th input across the ensemble members, i.e., $\{\texttt{MoE}(\mathbf{h}_{i,m})\}_{m=1}^{M}$, to be different. Indeed, without partitioning, each $\mathbf{h}_{i,m}$ could select K identical experts. As a result, we expect *Tiling* to output M similar predictions across ensemble members. This is confirmed in Table 5.2 where we observe that the KL for *Tiling* is orders of magnitude smaller than for E^3. To mitigate this effect, we also tried to increase the level of noise σ in $\texttt{gate}_K$ (by a factor $\{2, 4\}$), to cause the expert assignments to differ across $\{\mathbf{h}_{i,m}\}_{m=1}^{M}$. While we do see an increase in KL, *Tiling* still performs worse than E^3 across all metrics.

Table 5.3: ImageNet performance (mean ± std. err. over 8 seeds) of E^3-B/32 ($K = M = 2$), $E = 32$ total experts, and varying expert overlap between subsets $\mathcal{E}_m$.

Overlap	NLL ↓	Error ↓	ECE ↓	KL ↑
0 (=E^3)	**0.612** ±0.001	**16.49** ±0.02	**0.013** ±0.000	**0.198** ±0.003
2	0.617 ±0.003	16.55 ±0.09	0.016 ±0.001	0.167 ±0.005
4	0.622 ±0.001	16.62 ±0.02	0.017 ±0.001	0.148 ±0.003
8	0.627 ±0.002	16.67 ±0.07	0.021 ±0.001	0.122 ±0.010
16	0.639 ±0.004	16.82 ±0.07	0.030 ±0.003	0.077 ±0.011
32 (=Tiling)	0.637 ±0.002	16.74 ±0.06	0.028 ±0.001	0.000 ±0.000

Interestingly, we can interpret *Tiling* as an approximation, via M samples, of the marginalisation $\mathbb{E}_{\boldsymbol{\varepsilon}_1,\ldots,\boldsymbol{\varepsilon}_\ell}[f(\boldsymbol{x};\boldsymbol{\theta})]$ with respect to the noise $\{\boldsymbol{\varepsilon}_l\}_{l=1}^{\ell}$ in the ℓ MoE layers of $f(\cdot;\boldsymbol{\theta})$ (further assuming the capacity constraints of the experts, as described in Riquelme et al. (2021), do not bias the M samples).

Tiling with Increasing Parameter Sharing

The results in Table 5.2 (as well as Section C.8.3) suggest that the strong performance of E^3 is related to its high-diversity predictions. We hypothesise that this diversity is a result of the large number of non-shared parameters in each ensemble member, i.e., the partitioning of the experts. To test this hypothesis, we allow the subsets $\mathcal{E}_m$ in E^3 to have *some degree of overlap* (i.e., ensemble members share some experts), thus interpolating between E^3 and *Tiling*. For example, with total experts $E = 32$ and an ensemble size $M = 2$, an overlap of 8 shared experts means that each ensemble member has $(32 - 8)/2 = 12$ unique experts, and $12 + 8 = 20$ in total. Table 5.3 shows that increasing the number of shared experts directly decreases diversity and thus NLL, Error, and ECE. We see the same trends for $K = 1$ (rather than $K = 2$; see Table C.9).

Multiple Predictions without Tiling or Partitioning

As highlighted in Table 5.1, an ensemble of size M outputs M predictions for a given input (thereafter, averaged) while sparse MoEs only produce a single prediction. Thus, a natural question is how much the gains of E^3 are simply due to its ability to produce multiple predictions, rather than its specific tiling and partitioning mechanisms? To answer this, we investigate a simple multi-prediction variant of sparse MoEs (*Multi-pred*) wherein the last MoE layer of the form (5.1) is replaced by

$$\texttt{Multi-pred-MoE}(\mathbf{h}) = \{g_e(\mathbf{h}) \cdot \texttt{MLP}_e(\mathbf{h})\}_{g_e(\mathbf{h})>0} \in \mathbb{R}^{K \times Q}, \tag{5.3}$$

Table 5.4: ImageNet performance (mean ± std. err. over 8 seeds) of V-MoE-B/32 and a simple multi-prediction variant (*Multi-pred*) whose last MoE layer is changed as in (5.3).

	K	NLL ↓	Error ↓	ECE ↓	KL ↑
E^3 ($M = 2$)	1	**0.622** ±0.001	**16.70** ±0.03	**0.018** ±0.000	**0.217** ±0.003
Multi-pred	2	0.636 ±0.001	17.16 ±0.02	0.024 ±0.000	0.032 ±0.001
V-MoE	2	0.638 ±0.001	**16.76** ±0.05	0.033 ±0.001	—
E^3 ($M = 2$)	2	**0.612** ±0.001	**16.49** ±0.02	**0.013** ±0.000	**0.198** ±0.003
Multi-pred	4	0.645 ±0.001	17.39 ±0.04	0.021 ±0.000	0.011 ±0.001
V-MoE	4	0.636 ±0.001	16.70 ±0.04	0.034 ±0.001	—
E^3 ($M = 2$)	4	**0.611** ±0.001	**16.45** ±0.03	**0.014** ±0.000	**0.193** ±0.003
Multi-pred	8	0.650 ±0.001	17.50 ±0.03	0.021 ±0.000	0.005 ±0.000
V-MoE	8	0.635 ±0.002	16.72 ±0.06	0.028 ±0.001	—

where $\{g_e(\mathbf{h})\}_{e=1}^{E} = \texttt{gate}_K(\mathbf{Wh})$, and we have assumed $\texttt{MLP}_e(\mathbf{h}) \in \mathbb{R}^Q$. Instead of *summing* the expert outputs like in (5.1), we *stack* the K selected expert contributions (as a reminder, $\texttt{gate}_K$ zeroes out the $E - K$ smallest weights). Keeping track of those K contributions makes it possible to generate K different predictions per input as in the classifier of Figure 5.1, thus aiming at capturing model uncertainty around the true prediction.

Table 5.4 compares the ImageNet performance of this simple multiple-prediction method with the standard V-MoE and E^3. In all cases, *Multi-pred* under performs relative to E^3. Indeed, despite improvements in ECE, it is only for $K = 2$, that *Multi-pred* provides small gains in NLL relative to V-MoE, while its classification error is always worse. In fact, *Multi-pred* for $K = 4$ performs *worse* in terms of NLL, classification error, and diversity than for $K = 2$. The KL diversity metric indicates that the *Multi-pred* is unable to provide diverse predictions. This indicates that it is specifically tiling and partitioning in E^3 that provide good performance.

5.4.3 Comparison with other Efficient Ensembling Strategies

In the previous subsection, we saw that a simple approach to multiple predictions in a V-MoE model is unable to achieve good diversity in predictions and thus strong predictive performance. Following Havasi et al. (2020); Soflaei et al. (2020), a possible fix to this problem would be to have a multi-prediction *and multi-input* approach. Furthermore, other efficient ensembling strategies could provide alternative solutions to this problem. Unfortunately, as we show in Table 5.5, while common efficient ensembling strategies such as BE (Wen et al., 2020), Monte Carlo (MC) Dropout (Gal and Ghahramani, 2016),

Table 5.5: ImageNet performance (mean ± std. err. over 8 seeds) of different efficient ensemble approaches based on a ViT-B/32 architecture.

	K	M	NLL ↓	Error ↓	ECE ↓	KL ↑	GFLOPs ↓
ViT	–	–	0.688 ±0.003	18.65 ±0.08	0.022 ±0.000	—	**78.0**
BE ViT	–	2	0.682 ±0.003	18.47 ±0.05	0.021 ±0.000	0.040 ±0.001	97.1
V-MoE	2	–	0.638 ±0.001	**16.76** ±0.05	0.033 ±0.001	—	94.9
MC Dropout V-MoE	1	2	0.648 ±0.002	17.10 ±0.05	**0.019** ±0.001	0.046 ±0.000	97.2
MIMO V-MoE	2	2	0.636 ±0.002	16.97 ±0.04	0.028 ±0.001	0.000 ±0.000	96.3
	2	4	0.672 ±0.001	17.72 ±0.04	0.037 ±0.000	0.001 ±0.000	99.0
E^3	1	2	**0.622** ±0.001	**16.70** ±0.03	**0.018** ±0.000	**0.217** ±0.003	105.9

and Multi-input Multi-output (MIMO) (Havasi et al., 2020), do improve slightly on ViT/V-MoE, they are unable to match the performance of E^3. In Section C.7, we provide a detailed description of how we carefully apply these methods to ViT/V-MoE to ensure a fair evaluation (sometimes even designing extensions of these methods). We also give results for additional K and M values.

5.5 Evaluation

We now benchmark E^3 against V-MoE. As a baseline, we also include results for *downstream* ensembles of V-MoE and ViT. These ensembles offer a natural baseline against E^3 as they also use a single upstream checkpoint, are easy to implement, and provide consistent improvements upon V-MoE. In Section C.8.12, we compare with *upstream* ensembles that require multiple upstream checkpoints (Mustafa et al., 2020). All results correspond to the average over 8 (for {S, B, L} single models) or 5 (for H single models and all up/downstream ensembles) replications. In Section C.8 we provide results for additional datasets and metrics as well as standard errors. Following Riquelme et al. (2021), we compare the predictive-performance vs. compute cost trade-offs for each method across a range of ViT families. In the results below, E^3 uses $(K, M) = (1, 2)$, single V-MoE models use $K = 2$, V-MoE ensembles use $K = 1$, and all use $E = 32$. Experimental details, including those for our upstream training, downstream fine-tuning, hyperparameter sweeps, and (linear) few-shot evaluation, can be found in Section C.1.

V-MoE vs. ViT

- **Ensembles help V-MoE just as much as ViT.** Ensembling was expected to benefit ViT. However, Figures 5.3 to 5.8 suggest that ensembling provides

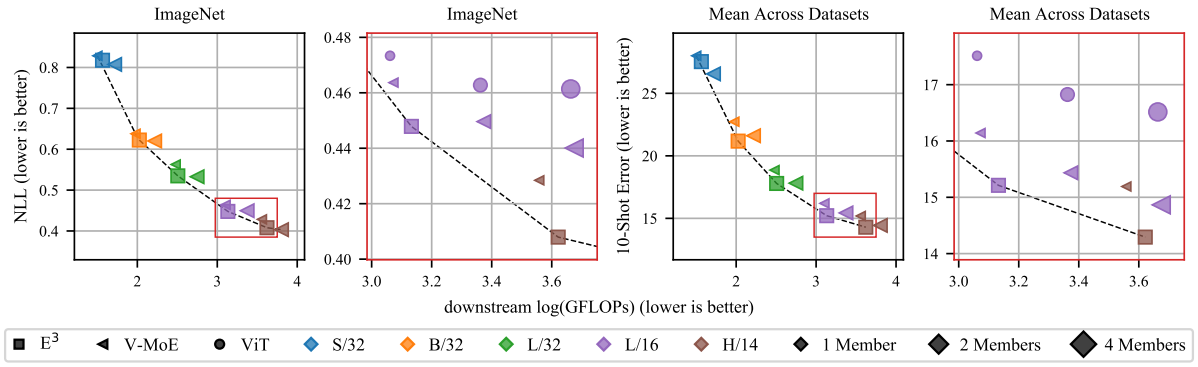


Figure 5.4: ImageNet NLL (left, centre left) and mean 10-shot error across datasets (centre right, right), with zoomed-in plots of highlighted areas. Zoomed-in plots include additional ensemble baselines. Dashed lines show Pareto frontiers, which tend to be defined by E^3.

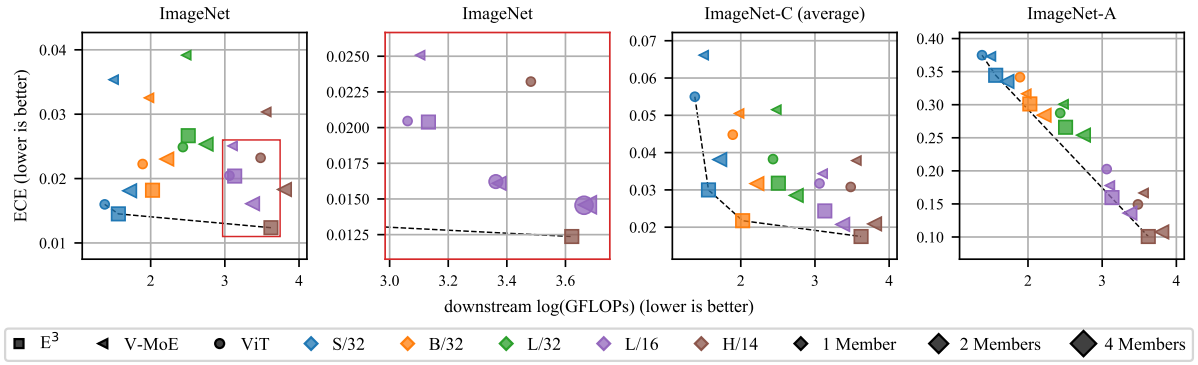


Figure 5.5: ECE for ImageNet (left), with a zoomed-in plot of the highlighted area (centre left), and under distribution shift (right, centre right). This is a metric for which ensembles are known to perform well, whereas, to the best of our knowledge, the performance of V-MoE has not been evaluated. The zoomed-in plot includes additional ensemble baselines for comparison. Dashed lines show Pareto frontiers.

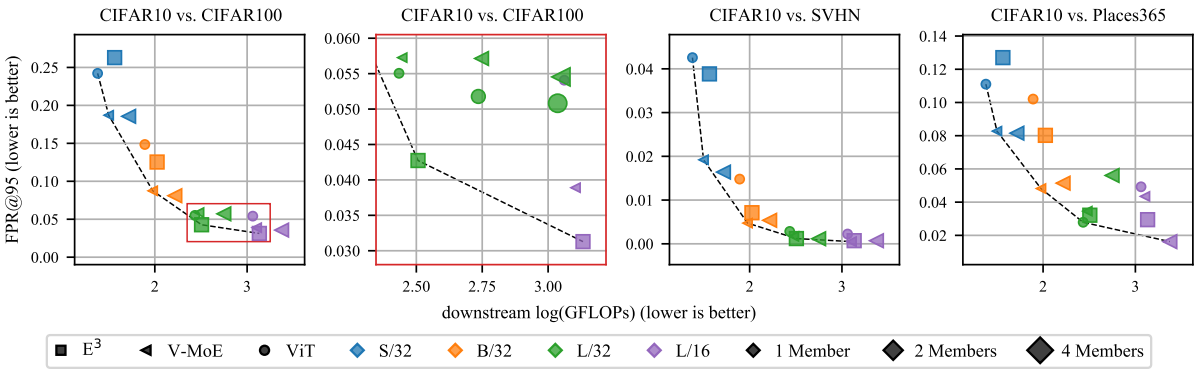


Figure 5.6: OOD detection, measured by false positive rate at 95% precision (Fort et al., 2021), for models fine-tuned on CIFAR10, with a zoomed-in plot of the highlighted area. This is a metric for which ensembles are known to perform well, whereas, to the best of our knowledge, the performance of V-MoE has not been evaluated. The zoomed-in plot includes additional ensemble baselines for comparison. Dashed lines show Pareto frontiers.

similar gains for V-MoE in terms of few-shot performance, NLL, ECE, OOD detection, and robustness to distribution shift. We believe this has not been observed before. Moreover, a downstream ensemble with four H/14 V-MoEs leads to an 88.8% accuracy on ImageNet (even reaching an impressive 89.3% for an upstream ensemble that further benefits from multiple upstream checkpoints, see Table C.12).

- **ECE is not consistent for different ViT/V-MoE families.** We see the ECE, unlike other metrics presented in this chapter, tends not to provide consistent trends as we increase the ViT family size (Figure 5.5). This observation is consistent with Minderer et al. (2021), who noted similar behaviour for a range of models. They note that post-hoc temperature scaling can improve consistency.

- **ViT consistently provides better ECE than V-MoE.** Surprisingly, despite V-MoE tending to have better NLL than ViT (Figure 5.3), their ECE is worse (Figure 5.5).

- **V-MoE outperforms ViT in OOD detection.** With L/32 being the only exception, V-MoE outperforms ViT on a range of OOD detection tasks (Figure 5.6). While this may seem surprising, given the opposite trend for ECE, it suggests that ViT makes more accurate predictions about the scale of the uncertainty estimates while V-MoE makes better predictions about the relative ordering of the uncertainty estimates.

- **For smaller ViT families, V-MoE outperforms ViT in the presence of distribution shift.** In contrast to the OOD detection results, Figure 5.7 shows that for smaller ViT families V-MoE improves on the performance of ViT. However, as the ViT family becomes larger, this trend is less consistent.

Efficient Ensemble of Experts

- **E^3 improves classification performance.** As shown in Figure 5.4, E^3 is either on or very near to the Pareto frontiers for NLL and 10-shot classification error, despite the fact that these are metrics for which ensembles and V-MoE, respectively, are known to perform well. Figure 5.8 shows that similar conclusions hold for CIFAR10/100 NLL.

- **E^3 performs best at the largest scale.** The difference in predictive performance between E^3 and V-MoE—or ensembles thereof—increases as the ViT family becomes

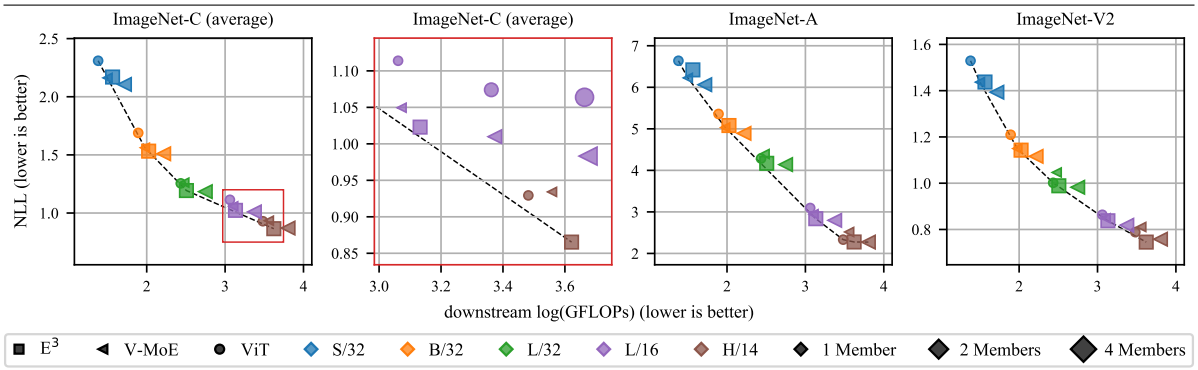


Figure 5.7: NLL under distribution shift for models trained on ImageNet. For ImageNet-C, we provide a zoomed-in plot of the highlighted area. The zoomed-in plot includes additional ensemble baselines. Dashed lines show Pareto frontiers, which tend to be defined by E^3.

larger (Figures 5.4 to 5.8, and Section C.4 where we propose a scheme to normalise performances across scales).

- **E^3 becomes Pareto efficient for larger ViT families in the presence of distribution shift.** Figure 5.7 shows that E^3 is more robust to distribution shift for larger ViT families, despite less consistent V-MoE performance at scale. When averaged over the shifted datasets (ImageNet-C, ImageNet-A, ImageNet-V2), E^3 improves on V-MoE in terms of NLL for all ViT families other than S/32, with improvements up to 8.33% at the largest scale; see Section C.8.10.

- **E^3 improves ECE over ViT and V-MoE.** Despite V-MoE providing poor ECE, E^3 does not suffer from this limitation (Figure 5.5). For most ViT families, E^3 also provides better ECE than V-MoE ensembles.

- **E^3 does not provide consistent OOD detection performance.** Firstly, Figure 5.6 shows that for small ViT families, E^3 performs worse than V-MoE and (even ViT in some cases). Nevertheless, as above, the relative performance improves for larger ViT families such that E^3 becomes Pareto efficient for two dataset pairs. Secondly, E^3 seems to perform better on the more difficult near OOD detection task (CIFAR10 vs. CIFAR100). These results, although sometimes subtle, are consistent across OOD detection metrics and dataset pairs, as shown in Section C.8.

Summary. While no single model performs best in all of our evaluation settings, we do find that E^3 performs well and is Pareto efficient in *most* cases. This is particularly true for the larger ViT families. One exception was OOD detection, where E^3 performance was somewhat inconsistent. On the other hand, while V-MoE clearly outperforms ViT in

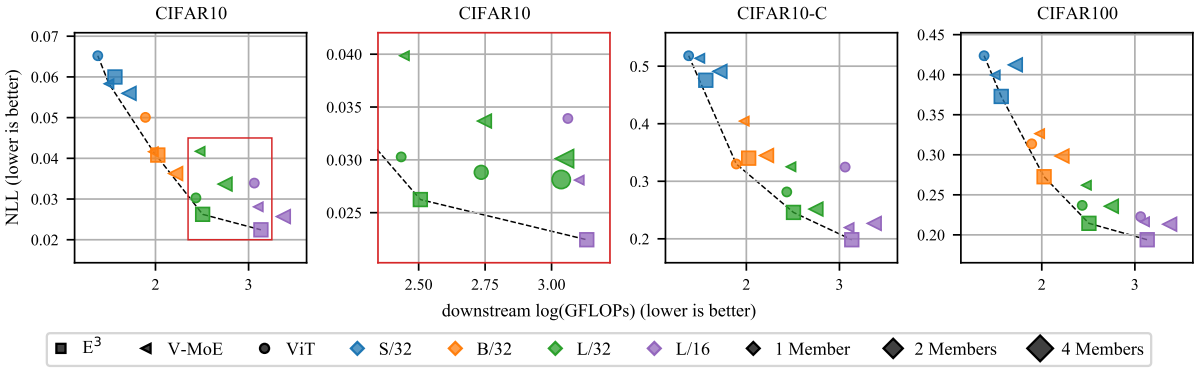


Figure 5.8: NLL for CIFAR10 (left), with a zoomed-in plot of the highlighted area (centre left), CIFAR10-C (centre right), and CIFAR100 (right). The zoomed-in plot includes additional ensemble baselines for comparison. Dashed lines show Pareto frontiers.

terms of accuracy, the uncertainty estimates can be better or worse depending on the downstream application.

5.6 Related Work

Mixture of Experts. MoEs (Jacobs et al., 1991; Jordan and Jacobs, 1994; Chen et al., 1999; Yuksel et al., 2012; Eigen et al., 2014) combine the outputs of different submodels, or *experts*, in an input-dependent way. Sparse MoEs only select a few experts per input, enabling to greatly scale models while keeping the prediction time constant. Sparse MoEs have been used to build large language models (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022). Recently, sparse MoEs have been also successfully applied to vision problems (Riquelme et al., 2021; Yang et al., 2021; Lou et al., 2021; Xue et al., 2022). Our work builds on the V-MoE architecture proposed by Riquelme et al. (2021), which is based on the ViT (Dosovitskiy et al., 2021) and showed improved performance for the same computational cost as ViT. We explore the interplay between sparse MoEs and ensembles and show that V-MoEs benefit from ensembling, by improving their predictive performance and robustness. While previous work studied ViT's calibration and robustness (Minderer et al., 2021; Fort et al., 2021; Paul and Chen, 2022; Mao et al., 2022), we are the first to study the robustness of V-MoE models.

Ensembles. Ensemble methods combine several different models to improve generalisation and uncertainty estimation. Ensembles achieve the best performance when they are composed of diverse members that make complementary errors (Hansen and Salamon, 1990; Fort et al., 2019; Wenzel et al., 2020b; D'Angelo and Fortuin, 2021; Lopes et al., 2022). However, standard ensembles are inefficient since they consist of multiple models,

each of which can already be expensive. To reduce test time, Xie et al. (2013) and Hinton et al. (2015); Tran et al. (2020); Nam et al. (2021) use compression and distillation mechanisms, respectively. To reduce training time, ensembles can be constructed with cyclical learning-rate schedules to snapshot models along the training trajectory (Huang et al., 2017; Zhang et al., 2019). Our work builds on BE (Wen et al., 2020) where a *single* model encapsulates an ensemble of networks, a strategy also explored by Lee et al. (2015); Havasi et al. (2020); Antorán et al. (2020); Dusenberry et al. (2020a); Ramé et al. (2021). Wenzel et al. (2020b) extended BE to models with different hyperparameters.

Bayesian Neural Networks. Bayesian Neural Networks (BNNs) are an alternative approach to deep ensembles for uncertainty quantification in neural networks. In a BNN the weights are treated as random variables and the uncertainty in the weights is translated to uncertainty in predictions via marginalisation. However, because the weight posterior is intractable for NNs, approximation is required. Popular approximations include Variational Inference (VI) (Hinton and van Camp, 1993; Graves, 2011; Blundell et al., 2015), the Laplace approximation (MacKay, 1992; Ritter et al., 2018), and MC Dropout (Gal and Ghahramani, 2016). However, many of these approximations make restrictive mean-field assumptions, which hurt performance (Foong et al., 2020; Coker et al., 2022; Fortuin et al., 2022b). Unfortunately, modelling full weight correlations for even small ResNets—with relatively few parameters compared to the ViT models considered here—is intractable (Daxberger et al., 2021b).

5.7 Summary

Our study of the interplay between sparse MoEs and ensembles has shown that these two classes of models are complementary. Efficient ensemble of experts exemplifies those mutual benefits—as illustrated by its accuracy, NLL, few-shot learning, robustness, and uncertainty calibration improvements over several challenging baselines in a range of benchmarks. We have also provided the first, to the best of our knowledge, investigation into the robustness and uncertainty calibration properties of V-MoEs—showing that these models are robust to dataset shift and are able to detect OOD examples.

In the next chapter, we will move away from the problem of uncertainty estimation in discriminative models and instead tackle data efficiency for deep generative models. We will approach this problem by building stronger inductive biases—specifically, about the symmetry transformations present in the data—into our models.

Chapter 6

A Generative Model of Symmetry Transformations

In this chapter, we are interested in improving the poor data efficiency of deep generative models. To this end, we propose a generative model that explicitly encodes the (partial) symmetries in the data. Our contributions are:

1. In Section 6.2, we propose a Symmetry-aware Generative Model (SGM). The SGM's latent representation is separated into an invariant component $\hat{\mathbf{x}}$ and an equivariant component $\boldsymbol{\eta}$. The latter, $\boldsymbol{\eta}$, captures the symmetries in the data, while $\hat{\mathbf{x}}$ captures none. We recover $\mathbf{x}$ by applying a parameterised transformation, $\mathbf{x} = \mathcal{T}_{\boldsymbol{\eta}}(\hat{\mathbf{x}})$. We call $\hat{\mathbf{x}}$ a *prototype* since each $\hat{\mathbf{x}}$ can produce arbitrarily transformed observations; see Figure 6.1.

2. In Section 6.2.1, we propose a two-stage algorithm for learning our SGM: first learning $\hat{\mathbf{x}}$ using a self-supervised approach and then learning $\boldsymbol{\eta}$ via maximum likelihood. Importantly, this does not require modelling the distribution of prototypes $p(\hat{\mathbf{x}})$, allowing the procedure to remain tractable even for complex data.

3. We discuss several potential pitfalls of training our SGM in Section 6.3. This discussion includes important practical suggestions as well as intuition for several of our design decisions.

4. Section 6.4 provides experimental validation for our method. First, we verify experimentally that our SGM completely captures affine and colour symmetries. Then we demonstrate that the test Marginal Log Likelihood (MLL) of a Variational Autoencoder (VAE) can be improved by building in an SGM. Additionally, unlike the base VAE, explicitly modelling symmetries makes our model's performance

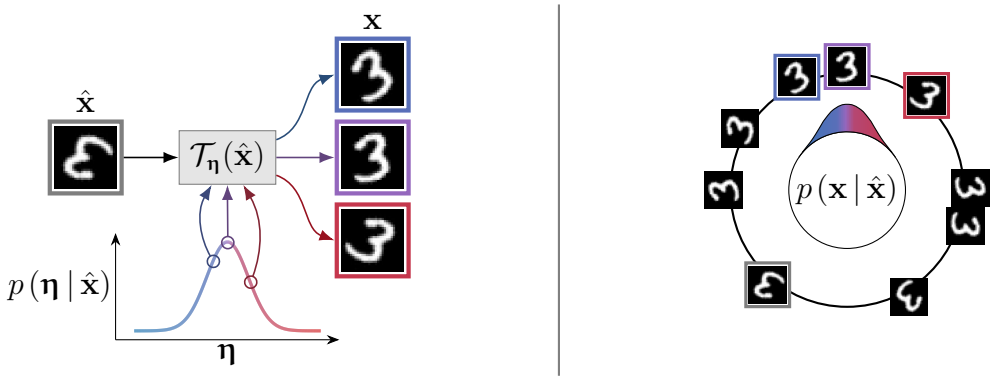


Figure 6.1: Left: An example of a symmetry-aware generative process which we aim to model in this chapter. A *prototype* $\hat{\mathbf{x}}$ (□) is transformed by $\mathcal{T}_\boldsymbol{\eta}$ into an observation $\mathbf{x}$ (□, □, □). The transformation—e.g., rotation—is parameterised by $\boldsymbol{\eta}$—e.g., an angle. **Right:** The corresponding orbit—i.e., the set of all possible instances of $\mathbf{x}$ that can result from applying $\mathcal{T}_\boldsymbol{\eta}$—with a few elements shown. Under this generative process, the prototype is an arbitrary orbit element. Each element in the orbit has a probability $p(\mathbf{x} \mid \hat{\mathbf{x}})$ induced by $p(\boldsymbol{\eta} \mid \hat{\mathbf{x}})$. E.g., for handwritten '3's, we expect digits in an upright orientation with some rotation around, say $\pm 40°$, corresponding to natural variations in handwriting.

robust to deleting half of the dataset (in an independent and identically distributed fashion).

This chapter is based on "A Generative Model of Symmetry Transformations" (Allingham et al., 2024). This paper was written with Bruno Mlodozeniec, Shreyas Padhy, Javier Antorán, David Krueger, Richard Turner, Eric Nalisnick, and José Miguel Hernández-Lobato. I led the project and was heavily involved with every aspect of the work, including ideation, exploration, coding, evaluation and presentation of the results, as well as writing the paper. David, Richard, Eric and Miguel gave high-level guidance and feedback. Shreyas and Bruno were instrumental in the coding and evaluation. Javier, Bruno, and Eric provided large contributions to the ideation and writing.

6.1 Motivation

Many physical phenomena exhibit symmetries; for example, many of the observable galaxies in the night sky share similar characteristics when accounting for their different rotations, velocities, and sizes. Hence, if we are to represent the world with generative models, they can be made more faithful and data-efficient by incorporating notions of symmetry. This has been well-understood for discriminative models for decades. Incorporating inductive biases such as invariance or equivariance to symmetry transformations dates back (at least) to ConvNets, which incorporate translation symmetries

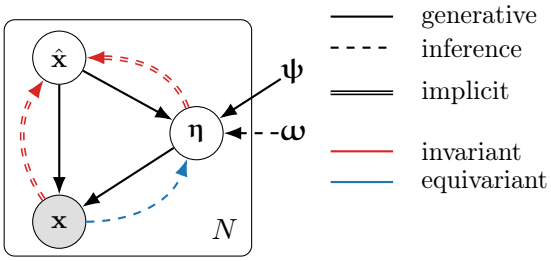


Figure 6.2: SGM graphical model. The *implicit* edges denote that $\hat{\mathbf{x}}$ is fully specified by $\boldsymbol{\eta}$ and $\mathbf{x}$—since $\hat{\mathbf{x}} = \mathcal{T}_{\boldsymbol{\eta}}^{-1}(\mathbf{x})$—and thus only $\boldsymbol{\eta}$ needs to be inferred given and observation $\mathbf{x}$.

(LeCun et al., 1989)—and can be extended to reflection and rotation (Cohen and Welling, 2016)—and more recently, transformers, with permutation symmetries (Lee et al., 2019).

In many cases, it is not known *a priori* which symmetries are present in the data. Learning symmetries in discriminative modelling is an active field of research (Nalisnick and Smyth, 2018; van der Wilk et al., 2018; Benton et al., 2020; Schwöbel et al., 2022; van der Ouderaa and van der Wilk, 2022; Rommel et al., 2022; Romero and Lohit, 2022; Immer et al., 2022, 2023; Miao et al., 2023; Mlodozeniec et al., 2023). However, in these works—which focus on invariant discriminative models—the label is often assumed to be invariant, and thus, the symmetry information can be *removed* rather than explicitly modelled. On the other hand, a generative model *must* capture the factors of variation corresponding to the symmetry transformations of the data. Doing so can provide benefits such as better representation learning—by disentangling symmetry from other latent variables (Antorán and Miguel, 2019)—and data efficiency—due to compact encoding of factor(s) of variation corresponding to symmetries. Furthermore, learning about underlying symmetries in data could be used for scientific discovery.

6.2 Symmetry-aware Generative Model (SGM)

Consider a dataset of observations $\{\boldsymbol{x}_n\}_{n=1}^N$ on a space $\mathcal{X}$, and a collection $\{\mathcal{T}_{\boldsymbol{\eta}}\}$ of transformations $\mathcal{T}_{\boldsymbol{\eta}} : \mathcal{X} \to \mathcal{X}$ parameterised by transformation parameters $\boldsymbol{\eta} \in \mathcal{H} \subseteq \mathbb{R}^{d_{\boldsymbol{\eta}}}$. We assume $\{\mathcal{T}_{\boldsymbol{\eta}}\}_{\boldsymbol{\eta}\in\mathcal{H}}$ (abbreviated $\{\mathcal{T}_{\boldsymbol{\eta}}\}$) form a group. Loosely, our aim is to model the distribution over transformations present in the data.

To do so, we model the distribution $p(\mathbf{x})$ by decomposing it into two disparate parts: **(1)** a distribution over prototypes and **(2)** a distribution over parameters controlling transformations to be applied to a prototype. Concretely, we specify our generative

model as follows (also depicted in Figure 6.2):

$$\hat{\mathbf{x}} \sim p(\hat{\mathbf{x}}), \tag{6.1}$$

$$\boldsymbol{\eta} \sim p_{\psi}(\boldsymbol{\eta} \mid \hat{\mathbf{x}}), \tag{6.2}$$

$$\mathbf{x} = \mathcal{T}_{\boldsymbol{\eta}}(\hat{\mathbf{x}}). \tag{6.3}$$

That is, the SGM assumes that each observation $\mathbf{x}$ is generated by applying a transformation $\mathcal{T}_{\boldsymbol{\eta}}$—parameterised by a latent variable $\boldsymbol{\eta}$—to a latent prototype $\hat{\mathbf{x}}$. Since $\hat{\mathbf{x}}$, by assumption, contains no information about the symmetries in the data, $p_{\psi}(\boldsymbol{\eta} \mid \hat{\mathbf{x}})$ must model the distribution over the transformations $\mathcal{T}_{\boldsymbol{\eta}}$ present in the data.

Why would we expect specifying $p(\mathbf{x})$ in this way to be useful? Firstly, our SGM allows us to query a distribution over naturally occurring transformations $p_{\psi}(\boldsymbol{\eta} \mid \hat{\mathbf{x}} = \mathcal{T}_{\boldsymbol{\eta}}^{-1}(\boldsymbol{x}))$ for any input $\boldsymbol{x}$, given the matching prototype $\hat{\boldsymbol{x}} := \mathcal{T}_{\boldsymbol{\eta}}^{-1}(\boldsymbol{x})$. Secondly, we expect our SGM to align with the true physical process of generating the data for many interesting datasets. As an illustrative example, when a person writes a digit, they first decide what kind of digit to write—e.g., the prototype could be an upright '3'—but when they put pen to paper, the digit they pictured is transformed due to various factors governing their handwriting[1]. Similarly, when a photographer captures an object, the photograph is also a function of latent factors of variation, such as lighting, the camera lens, camera shake, etc.

$\hat{\mathbf{x}}$ can informally be considered a canonical/reference example with no transformation applied to it. More precisely, we require that for any *orbit* of an element $\mathbf{x}$—defined as the set of elements in $\mathcal{X}$ which $\mathbf{x}$ can be mapped to by a transformation in $\{\mathcal{T}_{\boldsymbol{\eta}}\}$—there is exactly one prototype in the orbit. Figure 6.1 depicts an example orbit—a set {3, 3, 3, ...} of all rotated variants of a '3'—with a unique prototype.

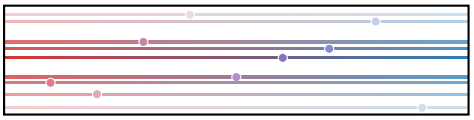


Figure 6.3: Orbits due to horizontal shift transformations. Each point (x_1, x_2) is transformed via $\mathcal{T}_{\eta} : (x_1, x_2) \mapsto (x_1, x_2) + (\eta, 0)$. Thus, horizontal lines form disjoint orbits in which any point can be transformed into any other point on the same line but not on another line. For each line, we can choose an arbitrary prototype (●) from which all other points on the line can be reached via $\mathcal{T}_{\eta}$.

Why do we want a group? Having the transformations $\{\mathcal{T}_{\boldsymbol{\eta}}\}$ be a group simplifies things, since $\{\mathcal{T}_{\boldsymbol{\eta}}\}$ will then naturally partition the space $\mathcal{X}$ into

[1]The alignment between our SGM and the data-generating process can be less clear-cut. E.g., a person is unlikely to "imagine" the same prototype for both a '6' or a '9'—two separate digits that can often be transformed into one another with a rotation.

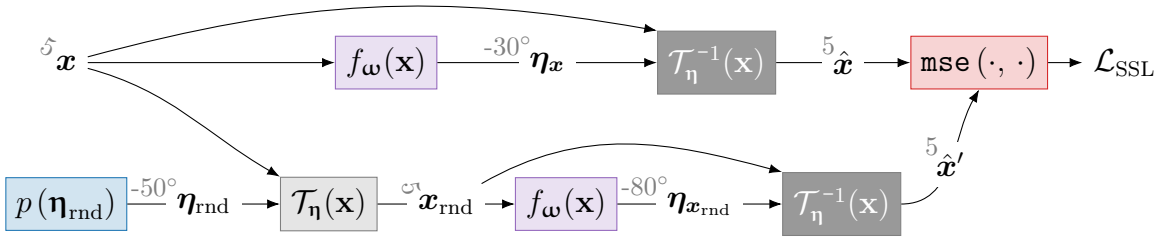


Figure 6.4: Self-supervised symmetry learning. We encourage $f_{\boldsymbol{\omega}}(\mathbf{x})$ to be equivariant by mapping $\boldsymbol{x}$ and a randomly transformed $\boldsymbol{x}$ to the same $\hat{\boldsymbol{x}}$. Gray text shows examples for each variable in the graph. Note that $\hat{\boldsymbol{x}}$ and $\boldsymbol{x}_{\text{rnd}}$ may not appear in the dataset; see Figure 6.1.

(disjoint) orbits. Within each orbit, every element can be transformed into one another with a transformation in $\{\mathcal{T}_{\boldsymbol{\eta}}\}$. As an example of such a partition, if our collection of transformations were horizontal shifts $\mathcal{T}_{\boldsymbol{\eta}} : \mathbf{x} \mapsto \mathbf{x} + (\boldsymbol{\eta}, 0)$ acting on a point $\mathbf{x} \in \mathbb{R}^2$, then the different orbits will correspond to all points on a given horizontal line; see Figure 6.3. Therefore, if we have chosen a unique prototype for each orbit and $\{\mathcal{T}_{\boldsymbol{\eta}}\}$ forms a group, any two elements $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ will have the same prototype if and only if they can be transformed into one another.

In Section 6.2.1, we describe a method for learning a transformation inference function $f_{\boldsymbol{\omega}} : \mathcal{X} \to \mathcal{H}$, with parameters $\boldsymbol{\omega}$, that for $\mathbf{x} \in \mathcal{X}$ returns transformation parameters $\boldsymbol{\eta} \in \mathcal{H}$ as $\boldsymbol{\eta} = f_{\boldsymbol{\omega}}(\mathbf{x})$. These map $\mathbf{x}$ to a prototype $\hat{\mathbf{x}} := \mathcal{T}_{\boldsymbol{\eta}}^{-1}(\mathbf{x})$ that generates $\mathbf{x} := \mathcal{T}_{\boldsymbol{\eta}}(\hat{\mathbf{x}})$[2]. We then apply standard generative modelling tools to learn $p(\hat{\mathbf{x}}, \boldsymbol{\eta}) = p(\hat{\mathbf{x}}) p_{\boldsymbol{\psi}}(\boldsymbol{\eta} \mid \hat{\mathbf{x}})$ given the generated data pairs $\{\hat{\boldsymbol{x}}_n, \boldsymbol{\eta}_n\}_{n=1}^N$.

6.2.1 Learning

We now discuss learning for the two NNs required by our model, starting with $f_{\boldsymbol{\omega}}(\mathbf{x})$ and then tackling $p_{\boldsymbol{\psi}}(\boldsymbol{\eta} \mid \hat{\mathbf{x}})$. In Section D.1 we discuss connections between our learning algorithm and MLL optimisation using an Evidence Lower BOund (ELBO).

Transformation inference function. For $\mathcal{T}_{\boldsymbol{\eta}}^{-1}$, with $\boldsymbol{\eta}$ given by $f_{\boldsymbol{\omega}}$, to map $\mathbf{x}$ to a prototype $\hat{\mathbf{x}}$, it must, by definition, map all elements in any given orbit to the same element in that orbit. In other words, the output of $\mathcal{T}_{f_{\boldsymbol{\omega}}(\boldsymbol{x})}^{-1}(\boldsymbol{x})$ should be *invariant* to transformations $\mathcal{T}_{\boldsymbol{\eta}'}$ of $\boldsymbol{x}$:

$$\mathcal{T}_{f_{\boldsymbol{\omega}}(\boldsymbol{x})}^{-1}(\boldsymbol{x}) = \mathcal{T}_{f_{\boldsymbol{\omega}}(\mathcal{T}_{\boldsymbol{\eta}'}(\boldsymbol{x}))}^{-1}\left(\mathcal{T}_{\boldsymbol{\eta}'}(\boldsymbol{x})\right), \quad \forall \boldsymbol{\eta}' \in \mathcal{H}. \tag{6.4}$$

[2]The transformation is not necessarily unique.

To learn such a function, we optimise for this property directly. To this end, we sample transformation parameters $\boldsymbol{\eta}_{\text{rnd}}$ from some distribution over parameters $p(\boldsymbol{\eta}_{\text{rnd}})$. This allows us to get random samples $\boldsymbol{x}_{\text{rnd}} := \mathcal{T}_{\boldsymbol{\eta}_{\text{rnd}}}(\boldsymbol{x}) \in \mathcal{X}$ in the orbit of any given element $\boldsymbol{x} \in \mathcal{X}$. Since we want full (i.e., strict) invariance, $p(\boldsymbol{\eta}_{\text{rnd}})$ must have support on the entire orbit (van der Ouderaa and van der Wilk, 2022). We then learn an equivariant $f_{\boldsymbol{\omega}}$ via a Self-Supervised Learning (SSL) scheme inspired by methods like BYOL (Grill et al., 2020) and, more directly, BINCE (Dubois et al., 2021). For example, we could use the objective illustrated in Figure 6.4:

$$\left\| \mathcal{T}^{-1}_{f_{\boldsymbol{\omega}}(\boldsymbol{x}_{\text{rnd}})}(\boldsymbol{x}_{\text{rnd}}) - \mathcal{T}^{-1}_{f_{\boldsymbol{\omega}}(\boldsymbol{x})}(\boldsymbol{x}) \right\|_2^2, \quad \boldsymbol{x}_{\text{rnd}} = \mathcal{T}_{\boldsymbol{\eta}_{\text{rnd}}}(\boldsymbol{x}),\ \boldsymbol{\eta}_{\text{rnd}} \sim p(\boldsymbol{\eta}_{\text{rnd}}). \tag{6.5}$$

Our actual objective differs slightly. Since $\mathcal{T}_{\boldsymbol{\eta}'}(\boldsymbol{x}') = \mathcal{T}_{\boldsymbol{\eta}''}(\boldsymbol{x}'')$ implies $\boldsymbol{x}' = \mathcal{T}^{-1}_{\boldsymbol{\eta}'} \circ \mathcal{T}_{\boldsymbol{\eta}''}(\boldsymbol{x}'')$, we instead use

$$\left\| \mathcal{T}_{f_{\boldsymbol{\omega}}(\boldsymbol{x})} \circ \mathcal{T}^{-1}_{f_{\boldsymbol{\omega}}(\boldsymbol{x}_{\text{rnd}})}(\boldsymbol{x}_{\text{rnd}}) - \boldsymbol{x} \right\|_2^2. \tag{6.6}$$

This change allows us to reduce the number of small discretisation errors introduced with each transformation application by replacing repeated transformations with a single composed transformation; see Section 6.3.1 for further discussion. Our SSL loss is given in Line 1 of Algorithm 1.

Generative model of transformations. Once we have a prototype inference function, we simply learn $p_{\boldsymbol{\psi}}(\boldsymbol{\eta} \mid \hat{\mathbf{x}})$ by maximum likelihood on the created data pairs $\left\{ f_{\boldsymbol{\omega}}(\boldsymbol{x}_i), \mathcal{T}^{-1}_{f_{\boldsymbol{\omega}}(\boldsymbol{x}_i)}(\boldsymbol{x}_i) \right\}$. This is shown in Line 8 of Algorithm 1. While we need to specify the kinds of symmetry transformations $\mathcal{T}_{\boldsymbol{\eta}}$ we expect to see in the data, by learning $p_{\boldsymbol{\psi}}(\boldsymbol{\eta} \mid \hat{\mathbf{x}})$ the model can learn the degree to which those transformations are present in the data. Thus, we can specify several potential symmetry transformations and learn that some are absent in the data. Furthermore, the required prior knowledge (the support of $p(\boldsymbol{\eta}_{\text{rnd}})$) is small compared to what our SGM can learn (the shapes of the distributions for each of the *present* transformations).

Since we are primarily interested in using the model to **(a)** inspect the distribution over naturally occurring transformations for a given element $\boldsymbol{x}$, and **(b)** resample new "naturally" augmented versions of the element, we *do not* need to learn $p(\hat{\mathbf{x}})$. We can do **(a)** by querying $p(\boldsymbol{\eta} \mid \hat{\mathbf{x}} = \hat{\boldsymbol{x}})$ for $\hat{\boldsymbol{x}} := \mathcal{T}^{-1}_{f_{\boldsymbol{\eta}}(\boldsymbol{x})}(\boldsymbol{x})$, and we can do **(b)** by sampling $\boldsymbol{\eta} \sim p(\boldsymbol{\eta} \mid \hat{\boldsymbol{x}})$ and transforming the $\hat{\boldsymbol{x}}$ to get $\boldsymbol{x} := \mathcal{T}_{\boldsymbol{\eta}}(\hat{\boldsymbol{x}})$. Of course, if one wanted to sample new prototypes, one could fit $p_{\boldsymbol{\theta}}(\hat{\mathbf{x}})$ using, e.g., a VAE or Normalising Flow (NF).

Algorithm 1 Learning

Require: initial parameters $\boldsymbol{\omega}_{\text{init}}$ & $\boldsymbol{\psi}_{\text{init}}$, dataset $\mathcal{D}$

function SSL_LOSS($\boldsymbol{x}, \boldsymbol{\omega}$)
  $\boldsymbol{\eta}_{\boldsymbol{x}} \leftarrow f_{\boldsymbol{\omega}}(\boldsymbol{x})$
  $\boldsymbol{\eta}_{\text{rnd}} \sim p(\boldsymbol{\eta}_{\text{rnd}})$
  $\boldsymbol{x}_{\text{rnd}} \leftarrow \mathcal{T}_{\boldsymbol{\eta}_{\text{rnd}}}(\boldsymbol{x})$
  $\boldsymbol{\eta}_{\boldsymbol{x}_{\text{rnd}}} \leftarrow f_{\boldsymbol{\omega}}(\boldsymbol{x}_{\text{rnd}})$
  $\boldsymbol{x}' \leftarrow \mathcal{T}_{\boldsymbol{\eta}_{\boldsymbol{x}}} \circ \mathcal{T}^{-1}_{\boldsymbol{\eta}_{\boldsymbol{x}_{\text{rnd}}}}(\boldsymbol{x}_{\text{rnd}})$
  output $\texttt{mse}(\boldsymbol{x}, \boldsymbol{x}')$
function MLE_LOSS($\boldsymbol{x}, \boldsymbol{\omega}, \boldsymbol{\psi}$)
  $\boldsymbol{\eta}_{\boldsymbol{x}} \leftarrow f_{\boldsymbol{\omega}}(\boldsymbol{x})$
  $\hat{\boldsymbol{x}} \leftarrow \mathcal{T}^{-1}_{\boldsymbol{\eta}_{\boldsymbol{x}}}(\boldsymbol{x})$
  output $-\log p_{\boldsymbol{\psi}}(\boldsymbol{\eta}_{\boldsymbol{x}} \mid \hat{\boldsymbol{x}})$
$\boldsymbol{\omega}, \boldsymbol{\psi} \leftarrow \boldsymbol{\omega}_{\text{init}}, \boldsymbol{\psi}_{\text{init}}$
while $\boldsymbol{\omega}$ not converged **do**
  $\boldsymbol{X} \leftarrow \texttt{next\_batch}(\mathcal{D})$
  update $\boldsymbol{\omega}$ with $\nabla_{\boldsymbol{\omega}} \frac{1}{B} \sum_{b=1}^{B} \text{SSL\_LOSS}(\boldsymbol{X}_b, \boldsymbol{\omega})$
while $\boldsymbol{\psi}$ not converged **do**
  $\boldsymbol{X} \leftarrow \texttt{next\_batch}(\mathcal{D})$
  update $\boldsymbol{\psi}$ with $\nabla_{\boldsymbol{\psi}} \frac{1}{B} \sum_{b} \text{MLE\_LOSS}(\boldsymbol{X}_b, \boldsymbol{\omega}, \boldsymbol{\psi})$
output $\boldsymbol{\omega}, \boldsymbol{\psi}$

Not needing to learn $p(\hat{\mathbf{x}})$ greatly simplifies training for complicated datasets that would otherwise require a large generative model, an observation made by Dubois et al. (2021).

6.3 Further Intuitions and Motivations

In this section, we begin by providing intuition for practical considerations involved with training our model. We then provide motivation for several of our modelling choices.

6.3.1 Practical Considerations

Training our SGM, while simple, has potential pitfalls in practice. We discuss the key considerations here and provide further recommendations in Section D.2.

Working with transformations. Repeated application of transformations—e.g., in Figure 6.4—can introduce unwanted artefacts such as blurring. For many useful transformations, we can often compose transformations before applying them to the input. For affine transformations of images, for example, we can directly multiply affine-

transformation matrices corresponding to the parameters $\boldsymbol{\eta}$. More generally, if there is some representation of the transformation parameters $T(\boldsymbol{\eta})$ where composition can be performed—e.g., as matrix multiplication $\mathcal{T}_{\boldsymbol{\eta}_2} \circ \mathcal{T}_{\boldsymbol{\eta}_1} = \mathcal{T}'_{T(\boldsymbol{\eta}_2)T(\boldsymbol{\eta}_1)}$, in the case where T is a group representation—then we recommend composing transformations in that space to minimize the number of applications. Additionally, NN architectures must be able to learn the equivariant mapping from $\mathbf{x}$ to $\boldsymbol{\eta}$. For example, using a standard CNN to represent a function that is (approximately) equivariant to continuous rotations would require many convolutional filters (Maile et al., 2023). Finally, one might notice that it is possible to remove the $\mathcal{T}_{\boldsymbol{\eta}}^{-1}$ operations from both paths of the SSL objective in Figure 6.4, and still have a valid objective (in $\mathcal{H}$-space rather than $\mathcal{X}$-space). However, the $\mathcal{X}$-space version is preferred since different parameters $\boldsymbol{\eta}_1, \boldsymbol{\eta}_2$ can map to the same transformed element $\mathcal{T}_{\boldsymbol{\eta}_1}(\boldsymbol{x}) = \mathcal{T}_{\boldsymbol{\eta}_2}(\boldsymbol{x})$. E.g., consider rotations transformations applied to a various shapes, for a square $\mathcal{T}_{0°} \equiv \mathcal{T}_{90°} \equiv \mathcal{T}_{180°} \equiv \mathcal{T}_{270°}$ all map to the same transformed image, and an $\mathcal{H}$-space objective incorrectly penalises differences of $\pm n \times 90°$ in $\boldsymbol{\eta}$ values.

Partial invertibility. In many common settings, transformations are not fully invertible. We encounter two such issues when working with affine transformations of images living in a finite, discrete coordinate space. Firstly, affine transformations are only *approximately* invertible in the discrete space due to the information loss when interpolating the transformed image onto a discrete grid. Thus, while only a single prototype $\hat{\mathbf{x}}$ exists for any $\mathbf{x}$, it may not be clear what the correct prototype is. Secondly, transformations can cause information loss due to the finite coordinate space (e.g., by shifting the contents of the image out-of-bounds[3]). We prevent significant information loss by augmenting the SSL loss in Line 1 of Algorithm 1 with an *invertibility loss*

$$\mathcal{L}_{\text{invertibility}}(\boldsymbol{\omega}) = \texttt{mse}\left(\boldsymbol{x}, \mathcal{T}^{-1}_{f_{\boldsymbol{\omega}}(\mathbf{x})}\left(\mathcal{T}_{f_{\boldsymbol{\omega}}(\mathbf{x})}(\boldsymbol{x})\right)\right). \tag{6.7}$$

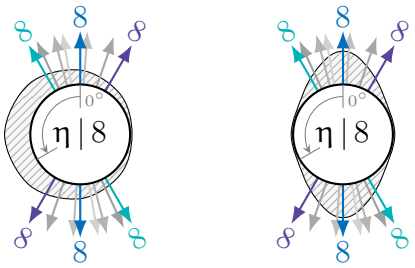
Instead, if appropriate bounds are known *a priori*, we can constrain $\boldsymbol{\eta}$ directly with `tanh`, `scale`, and `shift` bijectors.

Learning $p_{\boldsymbol{\psi}}(\boldsymbol{\eta} \mid \hat{\mathbf{x}})$ with imperfect inference. In practice, our transformation inference network $f_{\boldsymbol{\omega}}(\mathbf{x})$ will not be perfect; see Figure 6.10. Even after training, there may be small variations in the prototypes $\hat{\mathbf{x}}$ corresponding to different elements in the orbit of $\mathbf{x}$. To make $p_{\boldsymbol{\psi}}(\boldsymbol{\eta}_{\mathbf{x}} \mid \hat{\mathbf{x}})$ robust to these variations, we train it with prototypes corresponding to *randomly transformed* training datapoints. In other words, we implement the MLE

[3]This can occur in practice, since our SSL objective—which aims to make prototypes as similar as possible—can trivially be minimised by removing all of the contents of an image.

$\boldsymbol{x}$	$\hat{\boldsymbol{x}}$	$p(\eta \mid \boldsymbol{x}, \hat{\boldsymbol{x}})$
8	8	$0.5 \cdot \delta(\eta - 0°) + 0.5 \cdot \delta(\eta - 180°)$
♋	8	$0.5 \cdot \delta(\eta - 30°) + 0.5 \cdot \delta(\eta + 150°)$
♌	8	$0.5 \cdot \delta(\eta + 30°) + 0.5 \cdot \delta(\eta - 150°)$

(a) True distribution for η given $\mathbf{x}$ and $\hat{\mathbf{x}}$.



(b) Simple $p_\psi(\eta \mid \hat{\mathbf{x}})$ **(c)** Flexible $p_\psi(\eta \mid \hat{\mathbf{x}})$

Figure 6.5: Idealised examples of simple and flexible learned distributions over angles $p_\psi(\eta \mid \hat{\mathbf{x}})$— —given the true distribution $p(\eta \mid \hat{\mathbf{x}}) = \sum_{\mathbf{x} \in \{♋, \ldots, 8, \ldots, ♌\}} p(\eta \mid \mathbf{x}, \hat{\mathbf{x}})$— . Rotation is measured w.r.t. the positive y-axis.

objective in Line 8 of Algorithm 1 as $\log p_\psi(\boldsymbol{\eta}_{\boldsymbol{x}} \mid \hat{\boldsymbol{x}}')$, where $\hat{\boldsymbol{x}}' = \mathcal{T}^{-1}_{f_{\boldsymbol{\omega}}(\mathcal{T}_{\boldsymbol{\eta}_{\text{rnd}}}(\boldsymbol{x}))}(\mathcal{T}_{\boldsymbol{\eta}_{\text{rnd}}}(\boldsymbol{x}))$ as in our SSL objective. We find that averaging the loss over multiple samples—e.g., 5—of $\boldsymbol{\eta}_{\text{rnd}}$ is beneficial.

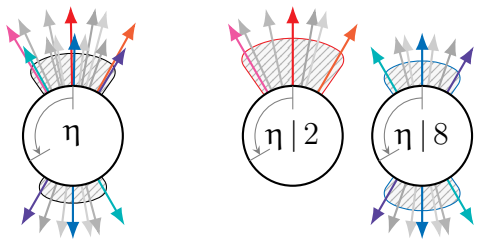
6.3.2 Modelling Choices

We now motivate some of the design choices for our SGM by means of illustrative examples. In each case, we assume that $\mathcal{T}_\eta$ is counter-clockwise rotation; thus, η is the angle.

1. The distribution $p_\psi(\eta \mid \hat{\mathbf{x}})$ is implemented as a NF. Consider a dataset of '8's rotated in the range $-30°$ to $30°$: $\{♋, \ldots, 8, \ldots, ♌\}$. Let us assume that the prototype is '8'. Figure 6.5a shows $p(\eta \mid \mathbf{x}, \hat{\mathbf{x}})$, the true distribution for η given $\mathbf{x}$ and $\hat{\mathbf{x}}$, for several observations, under the data generating process. These distributions are (sums of) deltas because only certain values of η will transform $\hat{\mathbf{x}}$ into $\mathbf{x}$. Because '8' is symmetric, $p(\eta \mid \mathbf{x}, \hat{\mathbf{x}})$ is the sum of two deltas. Figures 6.5b and 6.5c compare idealised examples of the learned $p_\psi(\eta \mid \hat{\mathbf{x}})$—given a *simple* uni-modal Gaussian family and a more *flexible* bi-modal mixture-of-Gaussian family—with the aggregate true distribution $p(\eta \mid \hat{\mathbf{x}}) = \sum_{\mathbf{x} \in \{♋, \ldots, 8, \ldots, ♌\}} p(\eta \mid \mathbf{x}, \hat{\mathbf{x}})$. Here, the simple unimodal distribution is clearly worse than the bimodal distribution due to the large amount of probability mass being wasted on angles with low density under the true data-generating process. Of course, one might argue that the bi-modal distribution is also not flexible enough. Furthermore, the definition of 'flexible enough' will be problem-specific. We solve this problem by using NFs, which can match a wide range of distributions.

$\boldsymbol{x}$	$\hat{\boldsymbol{x}}$	$p(\eta \mid \boldsymbol{x}, \hat{\boldsymbol{x}})$
2	2	$\delta(\eta - 0°)$
২	2	$\delta(\eta - 30°)$
২	2	$\delta(\eta + 30°)$
8	8	$0.5 \cdot \delta(\eta - 0°) + 0.5 \cdot \delta(\eta - 180°)$
৪	8	$0.5 \cdot \delta(\eta - 30°) + 0.5 \cdot \delta(\eta + 150°)$
৪	8	$0.5 \cdot \delta(\eta + 30°) + 0.5 \cdot \delta(\eta - 150°)$

(a) True distribution for η given $\mathbf{x}$ and $\hat{\mathbf{x}}$.



(b) η **(c)** $\eta \mid \hat{\mathbf{x}}$

Figure 6.6: Idealised examples of learned distributions over angles $p_\psi(\cdot)$— —with and without dependence on $\boldsymbol{\eta}$, given the true distribution $p(\cdot)$— .

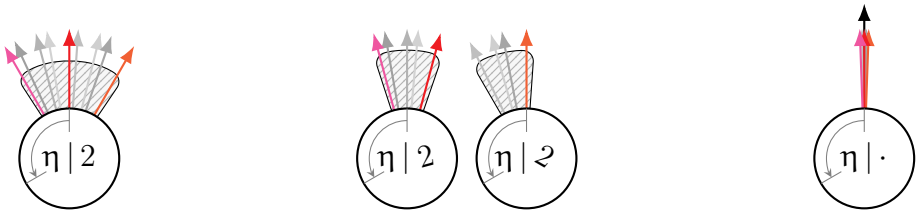
2. The transformation parameters η depend on the prototype $\hat{\mathbf{x}}$. Consider a dataset of '2's and '8's rotated in the range $-30°$ to $30°$: {২, …, 2, …, ২, ৪, …, 8, …, ৪}, with prototypes '2' and '8'. Figure 6.6a shows $p(\eta \mid \mathbf{x}, \hat{\mathbf{x}})$, the and example of a true distribution over η, for several observations. Figures 6.6b and 6.6c compare idealised examples of learned distributions over η and $\eta \mid \hat{\mathbf{x}}$. Without dependence on $\hat{\mathbf{x}}$, the model must place probability mass between $-150°$ and $150°$, in order to capture the symmetries of the '8's, however this results invalid digits—such as {৪, ৫, ৬}—which do not come from true data distribution. On the other hand, when η depends on $\hat{\mathbf{x}}$, this does not occur because the distribution conditioned on the prototype for the '2's only needs to place mass in $[-30°, 30°]$.

3. The prototype $\hat{\mathbf{x}}$ is *fully* invariant to transformations of $\mathbf{x}$. Models such as CNNs are most useful when we know *a priori* which symmetries are present in the data. However, in many cases, this must be learned. In the case of handwritten digit recognition, we know that the model should be invariant to some amount of rotation since people naturally write with some variation in angle. But a model that is invariant to rotations in the full range $[-180°, 180°]$ might be unable to distinguish between '6' and '9'. Thus, in the literature for learning invariances in the discriminative setting, it is common to learn *partially* invariant functions that capture some degree of invariance (van der Wilk et al., 2018; Benton et al., 2020; van der Ouderaa and van der Wilk, 2022). However, as we will now demonstrate, this approach is unsuitable for our SGM, as it breaks our assumption that $\hat{\mathbf{x}}$ contains no information about the symmetries in the data.

Consider a dataset of '2's rotated in the range $-30°$ to $30°$: {২, …, 2, …, ২}. Figure 6.7a shows predicted prototypes and the corresponding distributions over η for several observations. There are three cases: **(a)** a fully-invariant $\hat{\mathbf{x}}$, i.e., there is a single

	Full		Partial		None	
$\boldsymbol{x}$	$\hat{\boldsymbol{x}}$	$p(\eta \mid \boldsymbol{x}, \hat{\boldsymbol{x}})$	$\hat{\boldsymbol{x}}$	$p(\eta \mid \boldsymbol{x}, \hat{\boldsymbol{x}})$	$\hat{\boldsymbol{x}}$	$p(\eta \mid \boldsymbol{x}, \hat{\boldsymbol{x}})$
2	2	$\delta(\eta - 0°)$	2	$\delta(\eta + 15°)$	2	$\delta(\eta - 0°)$
2	2	$\delta(\eta - 30°)$	2	$\delta(\eta - 15°)$	2	$\delta(\eta - 0°)$
2	2	$\delta(\eta + 30°)$	2	$\delta(\eta - 0°)$	2	$\delta(\eta - 0°)$

(a) True $p(\eta \mid \hat{\mathbf{x}})$ with different degrees of invariance in $\mathbf{x}$.



(b) Full **(c)** Partial **(d)** None

Figure 6.7: Examples of learned distributions over angles $p_\psi(\eta \mid \hat{\mathbf{x}})$— ▨ / ↑ —with different amounts of invariance in the prototype $\hat{\mathbf{x}}$, given the true $p(\eta \mid \hat{\mathbf{x}})$— ↑↑↑↑ .

prototype, **(b)** a partially-invariant $\hat{\mathbf{x}}$, for which there are two prototypes in this example, and **(c)** a non-invariant $\hat{\mathbf{x}}$, which takes the partially-invariant case to the extreme and has as many prototypes as observations. In the partially-invariant and non-invariant cases, we can get multiple prototypes rather than a single unique prototype per orbit, which is invalid under the generative model of the data. As a result, $p_\psi(\boldsymbol{\eta} \mid \hat{\mathbf{x}})$ does not represent the distribution of naturally occurring transformations of $\hat{\mathbf{x}}$ in the data. This is illustrated in Figures 6.7b to 6.7d, which show idealized examples of the learned $p_\psi(\eta \mid \hat{\mathbf{x}})$ in each case. While the distribution in Figure 6.7b matches the distribution of transformations in the dataset, in Figures 6.7c and 6.7d we see that the distributions corresponding to non-unique prototypes do not.

To illustrate why this is a problem, let us say we would like to probe the probability of a particular transformed variant of an observed example. For example, given an example 3 of a digit '3', we want to know the probability of observing ε, that digit rotated by -90°. Assuming we can find a prototype $\hat{\boldsymbol{x}}$ we would like $p(\eta \mid \hat{\mathbf{x}} = \hat{\boldsymbol{x}})$ to represent all naturally occurring augmentations. Unless $\hat{\mathbf{x}}$ is unique, this won't necessarily be the case, as illustrated in Figure 6.7.

6.4 Experiments

In Section 6.4.1, we explore our SGM's understanding of the symmetries in a dataset. We show that it produces valid prototypes, and then given those prototypes that it generates

plausible samples from the data distribution. Then, in Section 6.4.2, we leverage our SGM to improve data-efficiency and robustness in deep generative models.

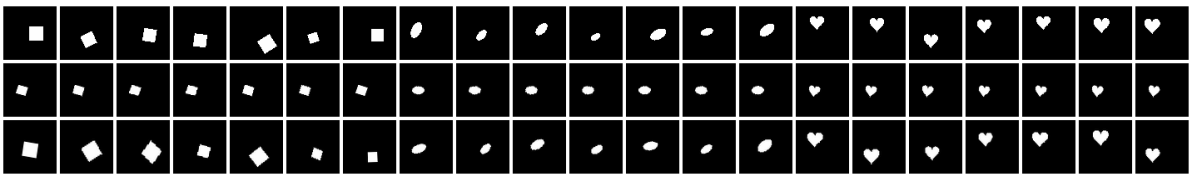
Here, we conduct experiments using three datasets—dSprites (Matthey et al., 2017), MNIST, and GalaxyMNIST (Walmsley et al., 2022)—and two kinds of transformations—affine and colour. Results for PatchCamelyon (Veeling et al., 2018) are in Section D.5. In Section 6.4.1, when working with MNIST under affine transformations, we add a small amount of rotation (in the range$[-15°, 15°]$) to the original data for visualisation purposes. For MNIST under colour transformations, we first convert the grey-scale images to colour images using only the red channel. We then add a random hue rotation in the range $[0, 0.6\pi]$ and a random saturation multiplier in the range $[0.6, 0.9]$. In the case of dSprites, we carefully control the rotations, positions, and sizes of all of the sprites. For example, in the case of the heart sprites, we have removed the rotations and set the y-positions to be bimodal in the top and bottom of the images. Further details about the dSprites setup, as well as all other experimental details, can be found in Section D.3.

We focus on learning affine transformations (shifting, rotation, and scaling) as they are expressive while still being a group that is easy to work with. We also learn colour transformations (hue, saturation, and value). See Section D.3.7 for details about how we parameterise $\mathcal{T}_{\boldsymbol{\eta}}$ in both cases.

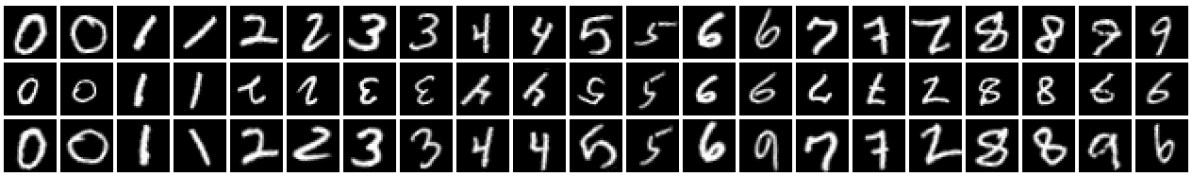
6.4.1 Learning Symmetries

Exploring transformations and prototypes. Figure 6.8 shows that for both datasets and kinds of transformations we consider, our SGM produces close-to-invariant prototypes as well as realistic "natural" examples that are almost indistinguishable from test examples. There are several illustrative examples which bear further discussion. The heart sprites in Figure 6.8a show that our SGM was able to learn *the absence* of a transformation (namely rotation) in the dataset. As expected, all of the prototypes for the sprites of the same shape are the same, since these shapes are in the same orbit as one another. This behaviour is also demonstrated for MNIST digits in Figures D.6 and D.7. The '6', '8', and '9' digits in Figure 6.8b demonstrate the ability of our SGM to learn bimodal distributions (on rotation in this case). The third *'7'* in the figure is interesting because our SGM has decided it looks more like a '2'.

Flexibility is important. In $\boldsymbol{\eta}$, each dimension corresponds to a different transformation. We refer to $p_{\boldsymbol{\psi}}(\boldsymbol{\eta}_i \mid \mathbf{x})$ as the marginal distribution of a single transformation parameter. Figure 6.9 shows these marginal learnt distributions for several digits from Figure 6.8b. We see that each of the parameters has its own range and shape. For

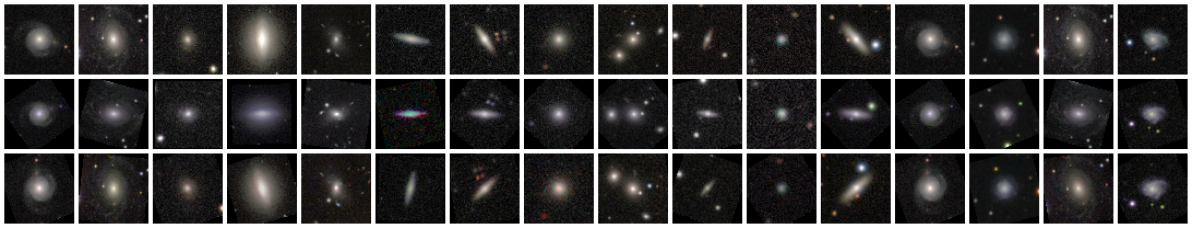


(a) dSprites under affine transformations



(b) MNIST under affine transformations

(c) MNIST under colour transformations



(d) GalaxyMNIST under affine and colour transformations

Figure 6.8: Top: samples from the test set. **Mid:** prototypes for each test example. **Bot:** resampled versions of each test example given the prototype. Prototypes for examples from the same orbit (and in some cases from distinct but similar orbits) match (e.g., their size, rotation, etc., are similar). Resampled examples are usually indistinguishable from test examples.

rotations, which are easy to reason about, we see distributions that make sense—the round '0' has an almost uniform distribution over rotations, the '1' and one of the '9's are strongly bimodal as expected. The other '9', which does not look as much like an upside-down '6', has a much smaller 2$^{\text{nd}}$ mode. The '2', which looks somewhat like an upside-down '7', is also bimodal. We see that prototypes of different sizes result in corresponding distributions over scaling parameters with different ranges. Figure D.8 provides additional examples for MNIST with colour transformations, and Figure D.9 investigates the distributions for dSprites. These results provide experimental evidence of the need for flexibility in the generative model for $p_\psi(\boldsymbol{\eta} \mid \mathbf{x})$, as conjectured in Section 6.3.2. We also find significant dependencies between dimensions of $\boldsymbol{\eta}$ (e.g., rotation and translation in dSprites).

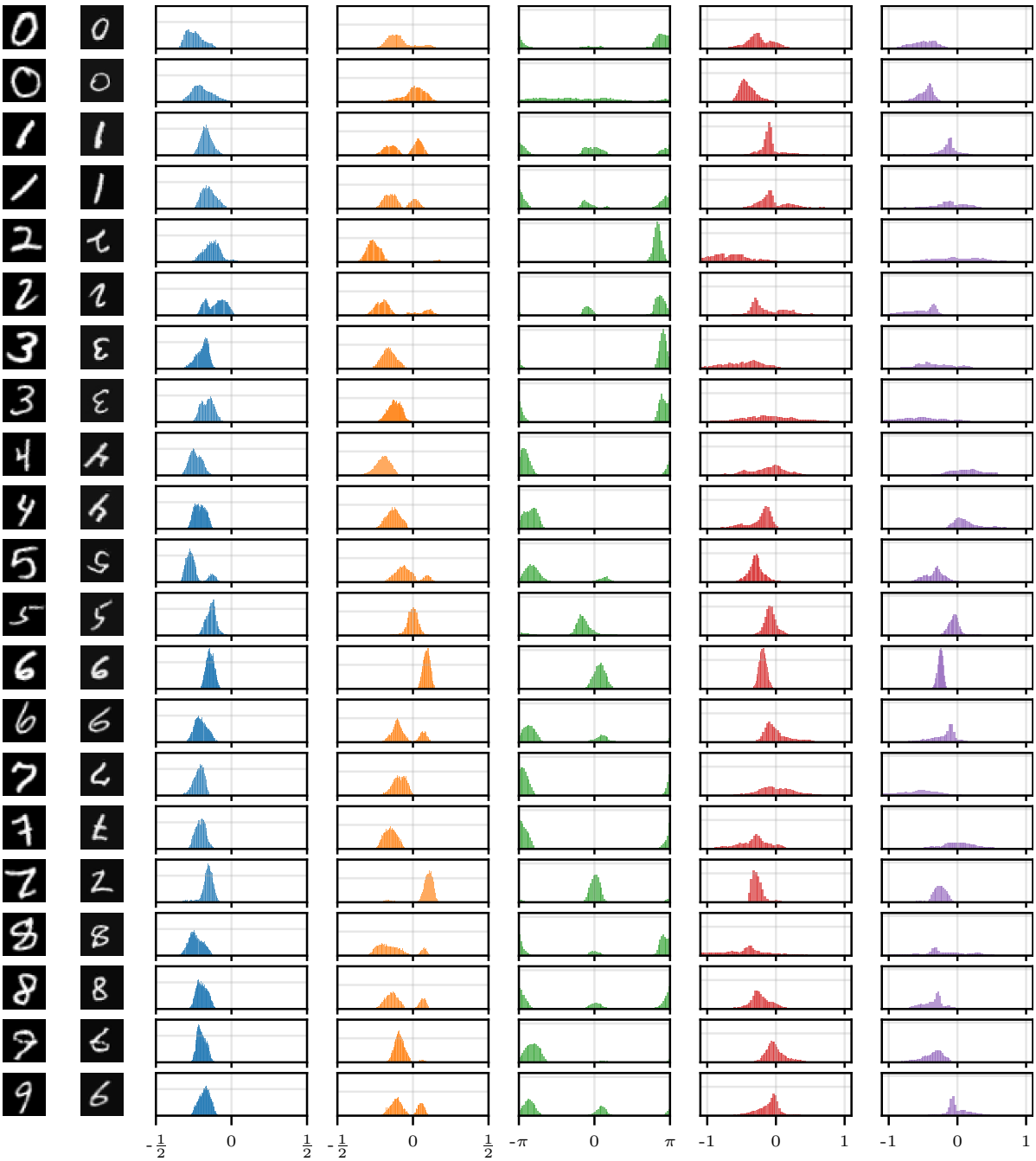


Figure 6.9: Left to right: test examples, their prototypes, and the corresponding marginals $p_\psi(\boldsymbol{\eta}_i \,|\, \mathbf{x})$ for translation in x, translation in y, rotation, scaling in x, and scaling in y.

Invariance of f_ω and the prototypes. In Figure 6.10, we investigate the imperfections of the inference network by considering an iterative procedure in which prototypes are treated as observed examples, allowing us to infer a chain of successive prototypes. We show several examples of such chains, as well as the average magnitude of the transformation parameters at each iteration, normalised by the maximum magnitude (at iteration 0). The first prototype $\hat{\mathbf{x}}_1$ is most different from the previous $\hat{\mathbf{x}}_0 = \mathbf{x}$, with successive prototypes being similar visually, and as measured by the magnitude of the inferred transformation parameters. However, the magnitude of the inferred parameters does not tend towards 0, but rather plateaus at around 5% of the maximum. This highlights that, although simple NNs can learn to be approximately invariant, there is potential to improve performance through the use of a natively invariant architecture.

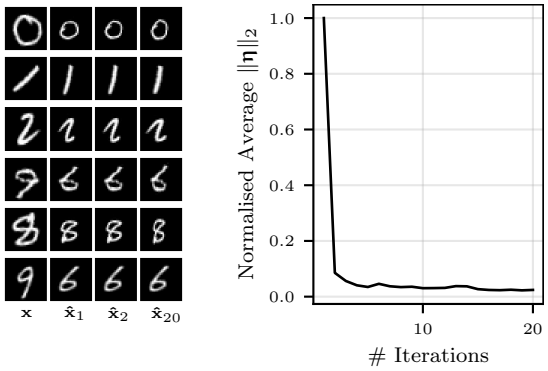


Figure 6.10: Iterative prototype inference. **Left:** starting with a test example $\mathbf{x}$, we get a prototype $\hat{\mathbf{x}}_1$, then treating prototype $\hat{\mathbf{x}}_i$ as an observed example we predict the next prototype $\hat{\mathbf{x}}_{i+1}$. **Right:** The average magnitude of the transformation parameters as a function of the number of iterations of this process.

6.4.2 VAE Data Efficiency

We use our SGM to build data-efficient and robust generative models. In Figure 6.11, we compare a standard VAE to two VAE-SGM hybrid models—AugVAE and InvVAE—for different amounts of training data and added rotation of the MNIST digits. When adding rotation, each $\boldsymbol{x}$ in the dataset set is always rotated by the same angle (sampled uniformly between $\pm\theta_{\text{max}}$, the maximum added rotation angle). Thus, adding rotation is *not* the same as data augmentation. AugVAE is a VAE that uses our SGM to re-sample transformed examples, introducing data augmentation at training time. InvVAE

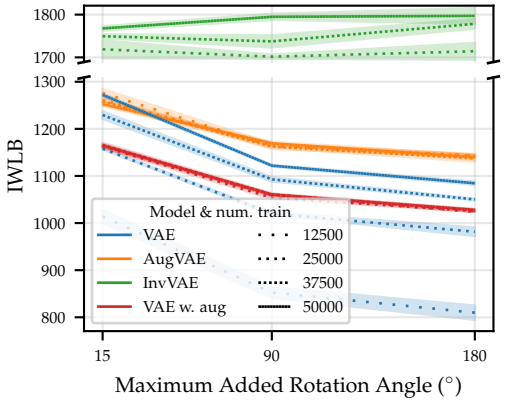


Figure 6.11: Incorporating symmetries improves data efficiency. Test-set Importance Weighted Lower Bound (IWLB) (mean and std. err. over 3 random seeds) on rotated MNIST for a standard VAE (w. and w.o. data aug.) and two VAE variants that incorporate symmetries via our SGM. Improved data efficiency is demonstrated by better performance with less training data and less sensitivity to added rotation.

is a VAE that uses our SGM to convert each example $\boldsymbol{x}$ to its prototype $\hat{\boldsymbol{x}}$ at both train and test time. That is, the VAE in InvVAE sees only the invariant representation of each example. We use test-set importance-weighted lower bound (IWLB) (Domke and Sheldon, 2018) of $p(\mathbf{x})$, estimated with 300 samples of the VAE's latent variable $\mathbf{z}$, and $\boldsymbol{\eta}$ for InvVAE, to compare the models. Further experimental details, such as hyperparameter sweeps, are in Section D.3.

As expected, for the VAE (▭), as we decrease the amount of training data (— → ····) or increase the amount of randomly added rotation, performance degrades. This is because the VAE sees fewer training examples *per-degree of rotation*. On the other hand, the AugVAE (▭) is more data efficient. Its performance is unaffected by reducing the number of observations by three-quarters. Furthermore, while the performance of AugVAE and the standard VAE are almost identical for small angles and large training sets, the drop in performance of AugVAE for larger random rotations is significantly smaller; AugVAE *does not* see fewer training examples *per-degree of rotation*. InvVAE (▭), which natively incorporates the inductive biases of our SGM, obtains a 500 nat larger likelihood than the other models. Its performance is almost perfectly robust to rotation in the dataset. Additionally, its metrics barely change ($< 10\%$) when trained on half the data. Finally, while the VAE with data augmentation (▭) improves on the standard VAE for less training data, it is substantially worse in the presence of more data. This contrasts with our AugVAE, which is almost always better. This poor performance is because the augmentations are independent of the samples. Thus, highly rotated digits can be rotated too much, smaller digits become too small, and digits near the image edges are moved out of frame. This highlights the importance of augmenting data in accordance with the true data distribution.

We further validate these results with the more complex GalaxyMNIST dataset and an enlarged set of both affine and colour transformations. As with our rotated MNIST with affine transformation results, in Figure 6.12, we see that AugVAE (▭) outperforms the standard VAE (▭). Furthermore, we see that AugVAE is robust to training with only half of the dataset. Our SGM captures the true data distribution with only 3500 training examples.

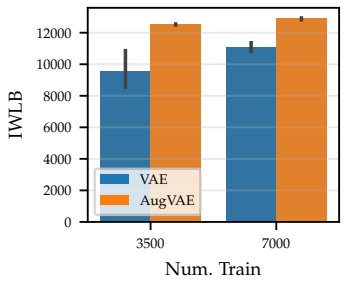


Figure 6.12: GalaxyMNIST data-efficiency (3 seed mean & std. err.).

6.5 Related Work

Learning Lie groups. Rao and Ruderman (1998); Miao and Rao (2007); Keurti et al. (2023) learn Lie groups from sequences of transformed images in an unsupervised fashion. Hashimoto et al. (2017) learn to represent an image as a linear combination of transformed versions of its nearest neighbours. Dehmamy et al. (2021) use Lie algebras to define CNNs for automatic symmetry discovery. Yang et al. (2023) use a GAN-based approach to learn transformations of examples that leave the original data distribution unchanged, thereby fooling a discriminator. Falorsi et al. (2019) introduce a reparameterisation trick for learning densities on arbitrary, but known, Lie groups. Chau et al. (2022) learn a generative model over Lie group transformations applied to prototypical images that are themselves composed of sparse combinations of learned dictionary elements.

Learning a prototype. Kaba et al. (2023) note that symmetry-based NNs are often constrained in their architectures. Like us, they propose to learn "canonicalization functions" that produce prototypical representations of the data. Mondal et al. (2023) show that such canonicalization functions can be used to make large-pre-trained NNs equivariant and, when combined with dataset-dependent symmetry priors, do not degrade performance. Similarly, Kim et al. (2023) learn architecture-agnostic equivariant functions by averaging a non-equivariant function over a probabilistic prototypical input. Finally, while not explicitly trained to produce prototypes, spatial transformer networks learn to undo transformations such as translation, scaling, and rotations (Jaderberg et al., 2015).

Data augmentations and symmetries. Prior work makes several connections between data augmentation and symmetries relevant to our findings. Bouchacourt et al. (2021b) show that invariances in the model tend to result from natural variations in the data rather than data augmentation or model architecture. This supports our approach of learning data augmentation from the data and our architecture-agnostic self-supervised invariance learning method. Balestriero et al. (2022); Miao et al. (2023); Bouchacourt et al. (2021b) show that learned symmetries (i.e., data augmentation) should be class-dependent, much like our transformations are prototype-dependent.

Symmetry-aware latent spaces. Encoding symmetries in latent space is well-studied. Higgins et al. (2018) posit that symmetry transformations that leave some parts of the world invariant are responsible for exploitable structure in any dataset. Thus, agents benefit from *disentangled* representations that separate out these transformations. Winter et al. (2022) split the latent space of an auto-encoder into invariant and equivariant

partitions. However, they rely on in- and equivariant NN architectures, contrasting with our self-supervised learning approach. Furthermore, they do not learn a generative model—they reconstruct the input exactly—thus, they cannot sample new observations given a prototype. Xu et al. (2021) propose group equivariant subsampling layers that allow them to construct autoencoders with equivariant representations. Shu et al. (2018) propose an autoencoder whose representations are split such that the reconstruction of an observation is decomposed into a "template" (much like our prototypes) and a spatial deformation (transformation).

In the generative setting, Louizos et al. (2016) construct a VAE with a latent space that is invariant to pre-specified sensitive attributes of the data. However, these sensitive attributes are observed rather than learned. Similarly, Aliee et al. (2023) construct a VAE with a partitioned latent space with a component that is invariant to spurious factors of variation in the data. Bouchacourt et al. (2018); Hosoya (2019) learn VAE with two latent spaces—a per-observation equivariant latent and an invariant latent shared across grouped examples. Other works have constructed rotation equivariant (Kuzina et al., 2022) and partitioned equivariant and invariant (Vadgama et al., 2022) latent spaces. Antorán and Miguel (2019); Ilse et al. (2020) split the latent space of a VAE into domain, class, and residual variation components. The first of which can capture rotation symmetry in handwritten digits. Unlike us, they require class labels and auxiliary classifiers. Keller and Welling (2021) construct a VAE with a topographically organised latent space such that an approximate equivariance is learned from sequences of observations. In contrast to the works above, Bouchacourt et al. (2021a) argue that learning symmetries should not be achieved via a partitioned latent space but rather by learning *equivariant operators* that are applied to the whole latent space. Finally, while Nalisnick and Smyth (2017) do not learn symmetries, their *information lower bound* objective is reminiscent of several works above—and our own, see Section D.1—in minimising the mutual information between two quantities when learning a prior.

Self-supervised Equivariant Learning Dangovski et al. (2022) generalise standard invariant SSL methods to produce representations that can be either insensitive (invariant) or sensitive (equivariant) to transformations in the data. Similarly, Eastwood et al. (2023) use a self-supervised learning approach to disentangle sources of variation in a dataset, thereby learning a representation that is equivariant to each of the sources while invariant to all others.

6.6 Summary

We have presented a Symmetry-aware Generative Model (SGM) and demonstrated that it is able to learn, in an unsupervised manner, a distribution over symmetries present in a dataset. This is done by modelling the observations as a random transformation of an invariant latent *prototype*. This is the first such model we are aware of. Building generative models that incorporate this understanding of symmetries results in significantly improved MLLs and robustness to data sparsity. This is exciting in the context of modern generative models, which are close to exhausting all of the data on the internet.

In the next, and final, chapter of this thesis, we will conclude by briefly summarising the main findings and contributions from each chapter, as well as providing suggestions for future work.

Chapter 7

Conclusion

In this thesis, we have asked the question, "How can probabilistic approaches be used to improve deep learning?" In particular, we have focused on two problems

1. uncertainty estimation for discriminative NNs (Chapters 3, 4 and 5), and
2. data-efficiency in deep generative models (Chapter 6).

Our first finding, presented in Chapter 3, is that Bayesian inference can be used to infer the depth of a neural network. This uncertainty over the depth of the NN can be translated into calibrated predictive uncertainty and improved robustness of the network. Depth uncertainty can also be applied to neural architecture search and active learning. However, we observed that providing high-quality uncertainty estimates relies on having an over-parameterised NN, since the network must be able to fit multiple diverse explanations of the data.

Secondly, in Chapter 4, we saw that weight-space Bayesian inference for modern neural networks *can* result in calibrated predictive uncertainty estimates—that are competitive with state-of-the-art baselines—despite the challenge of scaling to many weights. The key insight is that rather than applying poor approximations to *all* of the weights, one should instead apply expressive approximate inference to a small *subset* of the weights. In particular, we have shown that capturing correlations between weights is crucial to obtaining calibrated uncertainty estimates. Interestingly, crude approximations can be applied when selecting the subset. This observation suggests the importance of carefully considering how and where we apply approximations in our Bayesian inference schemes.

In Chapter 5, we showed that carefully implemented efficient ensembling schemes provide Pareto-efficient models on the uncertainty calibration vs. compute cost frontier of very large Mixture-of-Expert models. This is a potentially surprising observation

for two reasons. Firstly, there is no Bayesian inference involved—just marginalisation. Secondly, because ensembles and sparse Mixture-of-Expert models have several similar properties, it was not obvious that they are complementary when combined. However, we have provided experimental evidence suggesting that these two classes of models are indeed complementary. In doing so, we have also provided the first investigation into the uncertainty calibration and robustness of V-MoEs models.

Finally, in Chapter 6, we proposed a model whose generative process converts an invariant latent "prototype" into an observed example via a symmetry transformation. This generative model allows us to learn which symmetries are present in a dataset. We found that imbuing a standard deep generative model with this knowledge about the symmetries in the dataset leads to improved data-efficiency and importance-weighted lower bounds on the test-set marginal log-likelihood. This result suggests that discarding strong inductive biases about the data-generating process in favour of black-box deep generative models is not only unnecessary but can be harmful to performance.

7.1 Future Directions

In this section, we provide suggestions for further research on each of the topics explored in Chapters 3, 4, 5 and 6, combinations thereof, as well as the general area of probabilistic approaches for improving deep learning.

Depth Uncertainty. The lowest-hanging fruit for further work on depth uncertainty in NNs is its combination with traditional *weight* uncertainty. These techniques are orthogonal and should provide complementary benefits when combined. Next, we note that applying DUNs to CNNs is challenging due to the need for adaptation layers. However, the Transformer—which has become a ubiquitous NN architecture since DUNs were introduced—would not need such adaptation layers and is thus a natural candidate for depth uncertainty. Another, possibly more interesting, avenue for future work follows from the observation that DUN network weights have dual roles (1) fitting the data well, and (2) expressing diverse predictive functions at each depth. Thus, developing methods to ensure both roles are fulfilled could lead to large improvements.

Relatedly, we have seen that DUN performance is tied to excess model capacity; see the ImageNet results in Figure 3.13. This phenomenon also applies to MIMO ensembles (Havasi et al., 2020). These observations lead to two areas for further investigation. The first, more straightforward, direction is to apply DUNs to V-MoEs. The extra capacity provided by the experts might make it easier to learn diverse predictions at each depth.

Secondly, this suggests that a broad investigation into the role of overparameterisation in efficient ensemble performance could be fruitful.

Taking a step back, it seems that viewing DUNs through the lens of efficient ensembling techniques might be more pragmatic than the Bayesian perspective—since the number of hyperparameters is at least 6 orders of magnitude larger than the number of parameters being inferred, and from the Bayesian perspective this should cause overfitting which does not seem to happen in practice. Thus, we suggest this alternative perspective for those wishing to do further research into these models.

Subnetwork inference. While providing a general framework, our work on subnetwork inference was (possibly too) focused on the linearised Laplace method in practice. As a result, exploring subnetwork inference for other approximate inference schemes could readily provide new insights into the generality of the method and its interaction with different flavours of approximation. In particular, exploring full-covariance VI and HMC are natural candidates. The community has largely ignored the former for the same reasons that the full-covariance Laplace approximations are not used—storing covariance matrices is prohibitively expensive. Successfully—or unsuccessfully—combining another unimodal approximation with subnetwork inference could reveal properties about why subnetwork linearised Laplace, and the Laplace approximation more generally, work well. Furthermore, mean-field VI has several well-known pathologies (Foong et al., 2020; Coker et al., 2022). But these limitations do not apply to the setting with correlated weights, leaving the door open for subnetwork VI to perform well. HMC is an interesting candidate because, coupled with very large compute resources, it provides better-calibrated uncertainty estimates than other methods for inference in BNNs and could thus be used to better understand the impact of different sources of approximation in subnetwork inference.

Another potential avenue for investigation is a comparison with last-layer Laplace, which has emerged as one of the strongest and most practical variants of Laplace approximation since the subnetwork inference was introduced (Daxberger et al., 2021a). In particular, one could investigate *which* weights subnetwork inference tends to target and how much they overlap with the last layer of the NN. The Laplace approximation is not only useful due to its well-calibrated predictive uncertainty but also its computationally tractable model evidence, which is used for hyperparameter optimisation (Immer et al., 2021a; Antorán et al., 2022a,b; Immer et al., 2022). However, it is currently unclear whether or, more likely, to what extent subnetwork inference negatively impacts the estimation of the model evidence.

Finally, it would be interesting to investigate the combination of subnetwork inference and Kronecker-factorised covariance matrix approximations (Ritter et al., 2018; Martens and Grosse, 2015), which should allow the linearised Laplace approximation to scale to even larger NNs.

Sparse MoEs meet efficient ensembles. Our study of the interplay between sparse MoEs and ensembles involves two classes of models that are famously computationally expensive. As a result, there are two areas in which the results were limited and are thus ripe for future work. Firstly, while our study focused on downstream fine-tuned models, an extension to the upstream case and from-scratch training would also result in a fruitful investigation. In fact, in Section C.8.11, we provide some promising but preliminary results for from-scratch training. A complete study could investigate the impact of transfer learning with matching upstream and downstream efficient ensemble architectures as well as how well our findings hold for upstream combinations of deep ensembles and sparse MoEs.

Secondly, our study is exclusively focused on the computer vision domain. However, our results should be readily applicable to natural language modelling. With the growing prevalence of sparse MoEs in NLP (Patterson et al., 2021)—and the prevalence of NLP-based models in our daily lives—understanding and improving the robustness and reliability of such models becomes increasingly important. Our study makes steps in those directions and provide a solid starting point for future research.

Generative models of symmetries. The main limitation of our SGMs is that it requires specifying the super-set of possible symmetries. Future work might relax this requirement or explore how robust our SGMs is to even larger sets. Furthermore, care must sometimes be taken when specifying the set of symmetries. For example, when rotating images with "content" at the boundaries of the image; see Section D.5. This potential issue could also be further investigated in future work. Another obvious, but not to say unimportant, next step for this research direction is a broader set of evaluations. Applying our SGM to learning symmetries for more datasets and with more comprehensive classes of transformations would go a long way to demonstrating the generality of the method. Similarly, it would be useful to show that not only VAEs but also other generative models—such as NFs and diffusion models—benefit from incorporating symmetries. We are also excited about using SGMs for scientific discovery, given that the framework is ideal for probing naturally occurring symmetries present in the data. For example, we could apply SGMs to marginalise out the idiosyncrasies of different

measuring equipment and observation geometry in radio astronomy data. Additionally, given the success of using our SGM for data augmentation when training VAEs, it could be applied to data augmentation in the discriminative settings and compared with existing methods such as Benton et al. (2020); Immer et al. (2022); Miao et al. (2023).

Probabilistic approaches for improving deep learning. As we have seen in Chapter 4, applying inference in weight space becomes increasingly difficult as our models get larger, even with increasingly sophisticated approximate inference schemes. In Chapter 5, and arguably Chapter 3, we have seen that well-calibrated uncertainty estimates can be achieved via efficient ensembling techniques, especially for large overparameterised models. Thus, in today's LLM-dominated machine learning landscape, perhaps the probabilistic machine learning community should shy away from the traditional BNN approach of calibrated uncertainty estimation via inference in weight space. Clearly, the Bayesian approach is still practical for "smaller" models (even those as "large" as ResNets), and advances—e.g., in model selection and uncertainty quantification—can still be made there. However, there are many other areas in which the probabilistic approach can still naturally complement deep learning. Chapter 6 provided one example of building better inductive biases into deep generative models. The fields of active learning and experimental design provide many more examples. See Murray et al. (2021a,b) and Barbano et al. (2022); Antorán et al. (2023), for such examples in active learning and experimental design, respectively. Even simple uses of probabilistic thinking can be relevant. An example, in the setting of anytime algorithms, Jazbec et al. (2023) use a simple product-of-experts formulation to improve the anytime consistency of early-exit NNs. As a final example, Allingham et al. (2023) use likelihood ratios and ideas from outlier detection to do automatic prompt selection for zero-shot classifiers.

References

Jacob Abernethy, Chansoo Lee, and Ambuj Tewari. Perturbation techniques in online learning and optimization. *Perturbations, Optimization, and Statistics*, 2016. (Cited on p. 81.)

Hananeh Aliee, Ferdinand Kapl, Soroor Hediyeh-Zadeh, and Fabian J. Theis. Conditionally invariant representation learning for disentangling cellular heterogeneity. *arXiv preprint: 2307.00558*, 2023. (Cited on p. 114.)

James U Allingham. Unsupervised automatic dataset repair. Master's thesis, University of Cambridge, 2018. (Cited on p. vii.)

James Urquhart Allingham and Eric Nalisnick. A product of experts approach to early-exit ensembles. *1st ICML Workshop on Dynamic Neural Networks*, 2022. (Not cited.)

James Urquhart Allingham, Javier Antoran, Shreyas Padhy, Eric Nalisnick, and José Miguel Hernández-Lobato. Learning generative models with invariance to symmetries. *NeurIPS Workshop on Symmetry and Geometry in Neural Representations*, 2022a. (Cited on p. 211.)

James Urquhart Allingham, Florian Wenzel, Zelda E. Mariet, Basil Mustafa, Joan Puigcerver, Neil Houlsby, Ghassen Jerfel, Vincent Fortuin, Balaji Lakshminarayanan, Jasper Snoek, Dustin Tran, Carlos Riquelme Ruiz, and Rodolphe Jenatton. Sparse MoEs meet efficient ensembles. *Transactions on Machine Learning Research, TMLR*, 2022b. (Cited on pp. 3 and 78.)

James Urquhart Allingham, Jie Ren, Michael W Dusenberry, Xiuye Gu, Yin Cui, Dustin Tran, Jeremiah Zhe Liu, and Balaji Lakshminarayanan. A simple zero-shot prompt weighting technique to improve prompt ensembling in text-image models. In *Proceedings of the 39th International Conference on Machine Learning, ICML*, 2023. (Cited on p. 121.)

James Urquhart Allingham, Bruno Mlodozeniec, Shreyas Padhy, Javier Antorán, David Krueger, Richard Turner, Eric Nalisnick, and José Miguel Hernández-Lobato. A generative model of symmetry transformations. *Submitted to ICML*, 2024. (Cited on pp. 4 and 98.)

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul F. Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint: 1606.06565*, 2016. (Cited on pp. 26 and 54.)

Javier Antorán and Antonio Miguel. Disentangling and learning robust representations with natural clustering. In *18th IEEE International Conference On Machine Learning And Applications, ICMLA*, 2019. (Cited on pp. 99 and 114.)

Javier Antorán, James Urquhart Allingham, and José Miguel Hernández-Lobato. Depth uncertainty in neural networks. In *Advances in Neural Information Processing Systems 33, NeurIPS*, 2020. (Cited on pp. 3, 7, 25, 61, 65, 67, 69, 71, 74, 95, and 166.)

Javier Antorán, James Urquhart Allingham, and José Miguel Hernández-Lobato. Variational depth search in ResNets. *1st ICLR Workshop on Neural Architecture Search*, 2020. (Not cited.)

Javier Antorán, James Urquhart Allingham, David Janz, Erik Daxberger, Eric Nalisnick, and José Miguel Hernández-Lobato. Linearised Laplace inference in networks with normalisation layers and the neural g-prior. In *4th Symposium on Advances in Approximate Bayesian Inference, AABI*, 2022a. (Cited on p. 119.)

Javier Antorán, David Janz, James U Allingham, Erik Daxberger, Riccardo Rb Barbano, Eric Nalisnick, and José Miguel Hernández-Lobato. Adapting the linearised Laplace model evidence for modern deep learning. In *Proceedings of the 38th International Conference on Machine Learning, ICML*, 2022b. (Cited on p. 119.)

Javier Antorán, Riccardo Barbano, Johannes Leuschner, José Miguel Hernández-Lobato, and Bangti Jin. Uncertainty estimation for computed tomography with a linearised deep image prior. *Transactions on Machine Learning Research, TMLR*, 2023. (Cited on p. 121.)

Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *arXiv preprint: 1701.07875*, 2017. (Cited on p. 18.)

Arsenii Ashukha, Alexander Lyzhov, Dmitry Molchanov, and Dmitry P. Vetrov. Pitfalls of in-domain uncertainty estimation and ensembling in deep learning. In *8th International Conference on Learning Representations, ICLR*, 2020. (Cited on pp. 8, 17, 31, 50, 63, 67, 71, 156, 157, 165, and 166.)

Andrei Atanov, Arsenii Ashukha, Dmitry Molchanov, Kirill Neklyudov, and Dmitry P. Vetrov. Uncertainty estimation via stochastic batch normalization. In *16th International Symposium on Neural Networks, ISNN*, 2019a. (Cited on p. 12.)

Andrei Atanov, Alexandra Volokhova, Arsenii Ashukha, Ivan Sosnovik, and Dmitry P. Vetrov. Semi-conditional normalizing flows for semi-supervised learning. *arXiv preprint: 1905.00505*, 2019b. (Cited on p. 21.)

Randall Balestriero, Léon Bottou, and Yann LeCun. The effects of regularization and data augmentation are class dependent. In *Advances in Neural Information Processing Systems 35, NeurIPS*, 2022. (Cited on p. 113.)

Monika Bansal, Munish Kumar, Monika Sachdeva, and Ajay Mittal. Transfer learning for image classification using VGG19: Caltech-101 image data set. *Journal of Ambient Intelligence and Humanized Computing*, 2021. (Cited on p. 171.)

Riccardo Barbano, Johannes Leuschner, Javier Antorán, Bangti Jin, and José Miguel Hernández-Lobato. Bayesian experimental design for computed tomography with the linearised deep image prior. *arXiv preprint: 2207.05714*, 2022. (Cited on p. 121.)

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint: 1308.3432*, 2013. (Cited on pp. 78 and 80.)

Gregory W. Benton, Marc Finzi, Pavel Izmailov, and Andrew Gordon Wilson. Learning invariances in neural networks from training data. In *Advances in Neural Information Processing Systems 33, NeurIPS*, 2020. (Cited on pp. 99, 106, 121, 211, and 222.)

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *arXiv preprint: 1912.06680*, 2019. (Cited on p. 1.)

Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis R. Bach. Learning with differentiable perturbed optimizers. *arXiv preprint: 2002.08676*, 2020. (Cited on p. 81.)

Michael Betancourt. The fundamental incompatibility of scalable Hamiltonian Monte Carlo and naive data subsampling. In *Proceedings of the 31st International Conference on Machine Learning, ICML*, 2015. (Cited on pp. 12 and 73.)

Michael Betancourt. A conceptual introduction to Hamiltonian Monte Carlo. *arXiv preprint 1701.02434*, 2017. (Cited on p. 11.)

Umang Bhatt, Javier Antorán, Yunfeng Zhang, Q. Vera Liao, Prasanna Sattigeri, Riccardo Fogliato, Gabrielle Gauthier Melançon, Ranganath Krishnan, Jason Stanley, Omesh Tickoo, Lama Nachman, Rumi Chunara, Madhulika Srikumar, Adrian Weller, and Alice Xiang. Uncertainty as a form of transparency: Measuring, communicating, and using uncertainty. In *AAAI/ACM Conference on AI, Ethics, and Society, AIES*, 2021. (Cited on p. 54.)

Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006. (Cited on pp. 29, 54, and 59.)

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *Proceedings of the 31st International Conference on Machine Learning, ICML*, 2015. (Cited on pp. 10, 11, 31, 50, 56, 63, 73, and 95.)

Diane Bouchacourt, Ryota Tomioka, and Sebastian Nowozin. Multi-level variational autoencoder: Learning disentangled representations from grouped observations. In *The 32nd AAAI Conference on Artificial Intelligence, AAAI*, 2018. (Cited on p. 114.)

Diane Bouchacourt, Mark Ibrahim, and Stéphane Deny. Addressing the topological defects of disentanglement via distributed operators. *arXiv preprint: 2102.05623*, 2021a. (Cited on p. 114.)

Diane Bouchacourt, Mark Ibrahim, and Ari S. Morcos. Grounding inductive biases in natural images: invariance stems from variations in data. In *Advances in Neural Information Processing Systems 34, NeurIPS*, 2021b. (Cited on p. 113.)

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *7th International Conference on Learning Representations, ICLR*, 2019. (Cited on p. 18.)

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33, NeurIPS*, 2020. (Cited on pp. 1 and 78.)

Yuri Burda, Roger B. Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. In *4th International Conference on Learning Representations, ICLR*, 2016. (Cited on p. 22.)

David R. Burt, Sebastian W. Ober, Adrià Garriga-Alonso, and Mark van der Wilk. Understanding variational inference in function-space. In *3rd Symposium on Advances in Approximate Bayesian Inference, AABI*, 2021. (Cited on pp. 13, 26, 60, and 61.)

Ho Yin Chau, Frank Qiu, Yubei Chen, and Bruno A. Olshausen. Disentangling images with lie group transformations and sparse coding. *NeurIPS Workshop on Symmetry and Geometry in Neural Representations*, 2022. (Cited on p. 113.)

Ke Chen, Lei Xu, and Huisheng Chi. Improved learning algorithms for mixture of experts in multiclass classification. *Neural Networks*, 1999. (Cited on p. 94.)

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems 31, NeurIPS*, 2018. (Cited on p. 21.)

Tianqi Chen, Emily B. Fox, and Carlos Guestrin. Stochastic gradient Hamiltonian Monte Carlo. In *Proceedings of the 30th International Conference on Machine Learning, ICML*, 2014. (Cited on p. 11.)

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint 1710.09282*, 2017. (Cited on pp. 54, 57, and 63.)

M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi. Describing textures in the wild. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2014. (Cited on pp. 171, 173, and 188.)

Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical Japanese literature. *arXiv preprint: 1812.01718*, 2018. (Cited on p. 161.)

Taco Cohen and Max Welling. Group equivariant convolutional networks. In *Proceedings of the 32nd International Conference on Machine Learning, ICML*, 2016. (Cited on p. 99.)

Beau Coker, Wessel P. Bruinsma, David R. Burt, Weiwei Pan, and Finale Doshi-Velez. Wide mean-field Bayesian neural networks ignore the data. In *The 25th International Conference on Artificial Intelligence and Statistics, AISTATS*, 2022. (Cited on pp. 95 and 119.)

Chris Cremer, Xuechen Li, and David Duvenaud. Inference suboptimality in variational autoencoders. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, 2018. (Cited on p. 22.)

Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020. (Cited on p. 201.)

Francesco D'Angelo and Vincent Fortuin. Repulsive deep ensembles are Bayesian. In *Advances in Neural Information Processing Systems 34, NeurIPS*, 2021. (Cited on p. 94.)

Rumen Dangovski, Li Jing, Charlotte Loh, Seungwook Han, Akash Srivastava, Brian Cheung, Pulkit Agrawal, and Marin Soljacic. Equivariant self-supervised learning: Encouraging equivariance in representations. In *10th International Conference on Learning Representations, ICLR*, 2022. (Cited on p. 114.)

Erik Daxberger, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. Laplace redux - effortless Bayesian deep learning. In *Advances in Neural Information Processing Systems 34, NeurIPS*, 2021a. (Cited on pp. 73 and 119.)

Erik A. Daxberger, Eric T. Nalisnick, James Urquhart Allingham, Javier Antorán, and José Miguel Hernández-Lobato. Bayesian deep learning via subnetwork inference. In *Proceedings of the 37th International Conference on Machine Learning, ICML*, 2021b. (Cited on pp. 3, 54, and 95.)

Alexander G. de G. Matthews, Jiri Hron, Mark Rowland, Richard E. Turner, and Zoubin Ghahramani. Gaussian process behaviour in wide deep neural networks. In *6th International Conference on Learning Representations, ICLR*, 2018. (Cited on p. 12.)

Nima Dehmamy, Robin Walters, Yanchen Liu, Dashun Wang, and Rose Yu. Automatic symmetry discovery with lie algebra convolutional network. In *Advances in Neural Information Processing Systems 34, NeurIPS*, 2021. (Cited on p. 113.)

Marc Peter Deisenroth and Jun Wei Ng. Distributed Gaussian Processes. In *Proceedings of the 31st International Conference on Machine Learning, ICML*, 2015. (Cited on p. 35.)

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2009. (Cited on pp. 1, 73, 82, and 172.)

John S. Denker and Yann LeCun. Transforming neural-net output levels to probability distributions. In *Advances in Neural Information Processing Systems 3, NeurIPS*, 1990. (Cited on pp. 59 and 63.)

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT*, 2019. (Cited on p. 82.)

Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, 2000. (Cited on pp. 8 and 81.)

Georgi Dikov and Justin Bayer. Bayesian learning of neural network architectures. In *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS*, 2019. (Cited on pp. 14 and 50.)

Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: non-linear independent components estimation. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015. (Cited on pp. 19, 20, and 21.)

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *5th International Conference on Learning Representations, ICLR*, 2017. (Cited on pp. 13, 19, and 21.)

Justin Domke and Daniel Sheldon. Importance weighting and variational inference. In *Advances in Neural Information Processing Systems 31, NeurIPS*, 2018. (Cited on p. 112.)

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR*, 2021. (Cited on pp. 80, 82, 94, 169, 170, 171, and 183.)

Kevin Dowd. *Backtesting market risk models*. John Wiley & Sons, Ltd, 2013. (Cited on p. 36.)

Simon Duane, Anthony D. Kennedy, Brian J. Pendleton, and Duncan Roweth. Hybrid Monte Carlo. *Physics letters B*, 1987. (Cited on p. 11.)

Yann Dubois, Benjamin Bloem-Reddy, Karen Ullrich, and Chris J. Maddison. Lossy compression for lossless prediction. In *Advances in Neural Information Processing Systems 34, NeurIPS*, 2021. (Cited on pp. 102, 103, and 212.)

John C. Duchi, Peter L. Bartlett, and Martin J. Wainwright. Randomized smoothing for stochastic optimization. *SIAM J. Optim.*, 2012. (Cited on p. 81.)

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In *Advances in Neural Information Processing Systems 32, NeurIPS*, 2019. (Cited on pp. 21 and 215.)

Michael Dusenberry, Ghassen Jerfel, Yeming Wen, Yi-An Ma, Jasper Snoek, Katherine A. Heller, Balaji Lakshminarayanan, and Dustin Tran. Efficient and scalable Bayesian neural nets with rank-1 factors. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, 2020a. (Cited on pp. 13, 50, 74, 95, and 175.)

Michael W Dusenberry, Dustin Tran, Edward Choi, Jonas Kemp, Jeremy Nixon, Ghassen Jerfel, Katherine Heller, and Andrew M Dai. Analyzing the role of model uncertainty for electronic health records. In *Proceedings of the ACM Conference on Health, Inference, and Learning*, 2020b. (Cited on p. 78.)

Cian Eastwood, Julius von Kügelgen, Linus Ericsson, Diane Bouchacourt, Pascal Vincent, Bernhard Schölkopf, and Mark Ibrahim. Self-supervised disentanglement by leveraging structure in data augmentations. *arXiv preprint: 2311.08815*, 2023. (Cited on p. 114.)

David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. In *ICLR (Workshop Poster)*, 2014. (Cited on p. 94.)

Runa Eschenhagen, Erik Daxberger, Philipp Hennig, and Agustinus Kristiadi. Mixtures of Laplace approximations for improved post-hoc uncertainty in deep learning. *arXiv preprint: 2111.03577*, 2021. (Cited on p. 74.)

Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 2017. (Cited on p. 1.)

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: robust and efficient hyperparameter optimization at scale. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, 2018. (Cited on pp. 35 and 153.)

Luca Falorsi, Pim de Haan, Tim R. Davidson, and Patrick Forré. Reparameterizing distributions on lie groups. In *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS*, 2019. (Cited on p. 113.)

Sebastian Farquhar, Lewis Smith, and Yarin Gal. Liberty or depth: Deep Bayesian neural nets do not need complex weight posterior approximations. In *Advances in Neural Information Processing Systems 33, NeurIPS*, 2020. (Cited on p. 74.)

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research, JMLR*, 2022. (Cited on pp. 78 and 94.)

Angelos Filos, Sebastian Farquhar, Aidan N. Gomez, Tim G. J. Rudner, Zachary Kenton, Lewis Smith, Milad Alizadeh, Arnoud de Kroon, and Yarin Gal. A systematic comparison of Bayesian deep learning robustness in diabetic retinopathy tasks. *arXiv preprint: 1912.10481*, 2019. (Cited on pp. 26, 41, 72, 74, and 158.)

Andrew Y. K. Foong, Yingzhen Li, José Miguel Hernández-Lobato, and Richard E. Turner. 'In-between' uncertainty in Bayesian neural networks. *arXiv preprint: 1906.11537*, 2019. (Cited on pp. 34, 35, 56, 57, 58, 65, 66, 160, and 167.)

Andrew Y. K. Foong, David R. Burt, Yingzhen Li, and Richard E. Turner. On the expressiveness of approximate inference in Bayesian neural networks. In *Advances in Neural Information Processing Systems 33, NeurIPS*, 2020. (Cited on pp. 13, 26, 54, 56, 63, 74, 95, and 119.)

Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep ensembles: A loss landscape perspective. *arXiv preprint: 1912.02757*, 2019. (Cited on pp. 14, 54, 74, and 94.)

Stanislav Fort, Jie Ren, and Balaji Lakshminarayanan. Exploring the limits of out-of-distribution detection. In *Advances in Neural Information Processing Systems 34, NeurIPS*, 2021. (Cited on pp. 91 and 94.)

Vincent Fortuin, Mark Collier, Florian Wenzel, James Allingham, Jeremiah Liu, Dustin Tran, Balaji Lakshminarayanan, Jesse Berent, Rodolphe Jenatton, and Effrosyni Kokiopoulou. Deep classifiers with label noise modeling and distance awareness. *Transactions on Machine Learning Research, TMLR*, 2022a. (Not cited.)

Vincent Fortuin, Adrià Garriga-Alonso, Sebastian W. Ober, Florian Wenzel, Gunnar Rätsch, Richard E. Turner, Mark van der Wilk, and Laurence Aitchison. Bayesian neural network priors revisited. In *10th International Conference on Learning Representations, ICLR*, 2022b. (Cited on pp. 13 and 95.)

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR*, 2019. (Cited on pp. 56 and 74.)

Nicholas Frosst, Nicolas Papernot, and Geoffrey E. Hinton. Analyzing and improving representations with the soft nearest neighbor loss. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, 2019. (Cited on p. 28.)

Yarin Gal. *Uncertainty in deep learning*. PhD thesis, University of Cambridge, 2016. (Cited on p. 50.)

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 32nd International Conference on Machine Learning, ICML*, 2016. (Cited on pp. 12, 26, 31, 48, 50, 67, 73, 89, 95, 153, and 180.)

Jacob R. Gardner, Geoff Pleiss, Kilian Q. Weinberger, David Bindel, and Andrew Gordon Wilson. GPyTorch: Blackbox matrix-matrix Gaussian Process inference with GPU acceleration. In *Advances in Neural Information Processing Systems 31, NeurIPS*, 2018. (Cited on pp. 12 and 152.)

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P. Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of DNNs. In *Advances in Neural Information Processing Systems 31, NeurIPS*, 2018. (Cited on pp. 8 and 50.)

Adrià Garriga-Alonso, Carl Edward Rasmussen, and Laurence Aitchison. Deep convolutional networks as shallow Gaussian processes. In *7th International Conference on Learning Representations, ICLR*, 2019. (Cited on p. 12.)

Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 1992. (Cited on pp. 8 and 81.)

Gemini Team. Gemini: A family of highly capable multimodal models, 2023. (Cited on p. 1.)

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. MADE: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, 2015. (Cited on p. 17.)

Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 2015. (Cited on pp. 2 and 54.)

Zoubin Ghahramani and Michael I. Jordan. Supervised learning from incomplete data via an EM approach. In *Advances in Neural Information Processing Systems 7, NeurIPS*, 1993. (Cited on p. 2.)

Soumya Ghosh, Jiayu Yao, and Finale Doshi-Velez. Model selection in Bayesian neural networks via horseshoe priors. *Journal of Machine Learning Research, JMLR*, 2019. (Cited on pp. 14 and 50.)

M. N. Gibbs and D. J. C. MacKay. Efficient implementation of Gaussian processes for interpolation. 1996. (Cited on p. 12.)

Mark N. Gibbs. *Bayesian Gaussian processes for regression and classification*. PhD thesis, University of Cambridge, 1998. (Cited on p. 59.)

Clark R. Givens, Rae Michael Shortt, et al. A class of Wasserstein metrics for probability distributions. *The Michigan Mathematical Journal*, 1984. (Cited on p. 62.)

Tilmann Gneiting and Adrian E. Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American statistical Association*, 2007. (Cited on p. 16.)

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27, NeurIPS*, 2014. (Cited on p. 18.)

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016. (Cited on p. 60.)

Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training ImageNet in 1 hour. *arXiv preprint: 1706.02677*, 2017. (Cited on pp. 158 and 166.)

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. FFJORD: free-form continuous dynamics for scalable reversible generative models. In *7th International Conference on Learning Representations, ICLR*, 2019. (Cited on p. 21.)

Will Grathwohl, Kuan-Chieh Wang, Jörn-Henrik Jacobsen, David Duvenaud, Mohammad Norouzi, and Kevin Swersky. Your classifier is secretly an energy based model and you should treat it like one. In *8th International Conference on Learning Representations, ICLR*, 2020. (Cited on p. 18.)

Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems 24, NeurIPS*, 2011. (Cited on pp. 10, 11, 50, and 95.)

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Ávila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent - A new approach to self-supervised learning. In *Advances in Neural Information Processing Systems 33, NeurIPS*, 2020. (Cited on p. 102.)

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *Proceedings of the 33rd International Conference on Machine Learning, ICML*, 2017. (Cited on pp. 54 and 86.)

Fredrik K Gustafsson, Martin Danelljan, and Thomas B Schon. Evaluating scalable Bayesian deep learning methods for robust computer vision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020. (Cited on p. 79.)

Danijar Hafner, Dustin Tran, Timothy P. Lillicrap, Alex Irpan, and James Davidson. Noise contrastive priors for functional uncertainty. In *Proceedings of the 35th Conference on Uncertainty in Artificial Intelligence, UAI*, 2019. (Cited on pp. 13 and 50.)

Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 1990. (Cited on pp. 8, 81, and 94.)

Tatsunori B. Hashimoto, Percy Liang, and John C. Duchi. Unsupervised transformation learning via convex relaxations. In *Advances in Neural Information Processing Systems 30, NeurIPS*, 2017. (Cited on p. 113.)

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2017. (Cited on p. 171.)

Marton Havasi, Rodolphe Jenatton, Stanislav Fort, Jeremiah Zhe Liu, Jasper Snoek, Balaji Lakshminarayanan, Andrew Mingbo Dai, and Dustin Tran. Training independent subnetworks for robust prediction. In *8th International Conference on Learning Representations, ICLR*, 2020. (Cited on pp. 8, 89, 90, 95, 118, 180, 181, 182, 183, and 200.)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision, ICCV*, 2015. (Cited on pp. 153, 156, 157, and 166.)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2016a. (Cited on pp. 1, 40, 67, 157, and 165.)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, 2016b. (Cited on p. 159.)

Dan Hendrycks and Thomas G. Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *7th International Conference on Learning Representations, ICLR*, 2019. (Cited on pp. 41, 69, 158, 161, and 172.)

Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2021. (Cited on p. 172.)

James Hensman, Nicoló Fusi, and Neil D. Lawrence. Gaussian processes for big data. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence, UAI*, 2013. (Cited on pp. 12, 35, and 160.)

José Miguel Hernández-Lobato and Ryan P. Adams. Probabilistic backpropagation for scalable learning of Bayesian neural networks. In *Proceedings of the 31st International Conference on Machine Learning, ICML*, 2015. (Cited on pp. 10, 26, 35, 48, 50, 55, 66, 73, 153, 160, and 167.)

Irina Higgins, Loïc Matthey, Arka Pal, Christopher P. Burgess, Xavier Glorot, Matthew M. Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *5th International Conference on Learning Representations, ICLR*, 2017. (Cited on p. 22.)

Irina Higgins, David Amos, David Pfau, Sébastien Racanière, Loïc Matthey, Danilo J. Rezende, and Alexander Lerchner. Towards a definition of disentangled representations. *arXiv preprint: 1812.02230*, 2018. (Cited on p. 113.)

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *NeurIPS Deep Learning and Representation Learning Workshop*, 2015. (Cited on p. 95.)

Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Comput.*, 2002. (Cited on p. 17.)

Geoffrey E. Hinton and Drew van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the 6th Annual ACM Conference on Computational Learning Theory, COLT*, 1993. (Cited on pp. 10, 11, 50, and 95.)

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems 33, NeurIPS*, 2020. (Cited on pp. 18 and 19.)

Matthew D. Hoffman, David M. Blei, Chong Wang, and John W. Paisley. Stochastic variational inference. *Journal of Machine Learning Research, JMLR*, 2013. (Cited on p. 10.)

Haruo Hosoya. Group-based learning of disentangled representations with generalizability for novel contents. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI*, 2019. (Cited on p. 114.)

Neil Houlsby, Ferenc Huszar, Zoubin Ghahramani, and Máté Lengyel. Bayesian active learning for classification and preference learning. *arXiv preprint: 1112.5745*, 2011. (Cited on p. 48.)

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision, ECCV*, 2016. (Cited on pp. 31 and 50.)

Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. Snapshot ensembles: Train 1, get M for free. In *5th International Conference on Learning Representations, ICLR*, 2017. (Cited on pp. 8, 50, and 95.)

Maximilian Ilse, Jakub M. Tomczak, Christos Louizos, and Max Welling. DIVA: domain invariant variational autoencoders. In *International Conference on Medical Imaging with Deep Learning, MIDL 2020, 6-8 July 2020, Montréal, QC, Canada*, 2020. (Cited on p. 114.)

Alexander Immer, Matthias Bauer, Vincent Fortuin, Gunnar Rätsch, and Mohammad Emtiyaz Khan. Scalable marginal likelihood estimation for model selection in deep learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML*, 2021a. (Cited on p. 119.)

Alexander Immer, Maciej Korzepa, and Matthias Bauer. Improving predictions of Bayesian neural nets via local linearization. In *The 24th International Conference on Artificial Intelligence and Statistics, AISTATS*, 2021b. (Cited on pp. 57, 58, 59, and 73.)

Alexander Immer, Tycho F. A. van der Ouderaa, Gunnar Rätsch, Vincent Fortuin, and Mark van der Wilk. Invariance learning in deep neural networks with differentiable Laplace approximations. In *Advances in Neural Information Processing Systems 35, NeurIPS*, 2022. (Cited on pp. 99, 119, 121, and 211.)

Alexander Immer, Tycho F. A. van der Ouderaa, Mark van der Wilk, Gunnar Rätsch, and Bernhard Schölkopf. Stochastic marginal likelihood gradients using neural tangent kernels. In *Proceedings of the 39th International Conference on Machine Learning, ICML*, 2023. (Cited on p. 99.)

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 31st International Conference on Machine Learning, ICML*, 2015. (Cited on pp. 12, 159, and 160.)

Pavel Izmailov, Wesley J. Maddox, Polina Kirichenko, Timur Garipov, Dmitry P. Vetrov, and Andrew Gordon Wilson. Subspace inference for Bayesian deep learning. In *Proceedings of the 35th Conference on Uncertainty in Artificial Intelligence, UAI*, 2019. (Cited on pp. 12, 13, 33, 34, 54, 56, and 74.)

Pavel Izmailov, Polina Kirichenko, Marc Finzi, and Andrew Gordon Wilson. Semi-supervised learning with normalizing flows. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, 2020. (Cited on p. 21.)

Pavel Izmailov, Patrick Nicholson, Sanae Lotfi, and Andrew G Wilson. Dangers of Bayesian model averaging under covariate shift. *Advances in Neural Information Processing Systems 34, NeurIPS*, 2021a. (Cited on p. 13.)

Pavel Izmailov, Sharad Vikram, Matthew D. Hoffman, and Andrew Gordon Wilson. What are Bayesian neural network posteriors really like? In *Proceedings of the 37th International Conference on Machine Learning, ICML*, 2021b. (Cited on pp. 12, 13, 49, and 73.)

R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 1991. (Cited on p. 94.)

Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. In *Advances in Neural Information Processing Systems 28, NeurIPS*, 2015. (Cited on pp. 113 and 222.)

Metod Jazbec, James Urquhart Allingham, Dan Zhang, and Eric Nalisnick. Towards anytime classification in early-exit architectures by enforcing conditional monotonicity. In *Advances in Neural Information Processing Systems 36, NeurIPS*, 2023. (Cited on p. 121.)

Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural Computation*, 1994. (Cited on p. 94.)

Sékou-Oumar Kaba, Arnab Kumar Mondal, Yan Zhang, Yoshua Bengio, and Siamak Ravanbakhsh. Equivariance with learned canonicalization functions. In *Proceedings of the 39th International Conference on Machine Learning, ICML*, 2023. (Cited on p. 113.)

Jakob Nikolas Kather, Cleo-Aron Weis, Francesco Bianconi, Susanne M Melchers, Lothar R Schad, Timo Gaiser, Alexander Marx, and Frank Gerrit Zöllner. Multi-class texture analysis in colorectal cancer histology. *Scientific reports*, 2016. (Cited on p. 171.)

T. Anderson Keller and Max Welling. Topographic vaes learn equivariant capsules. In *Advances in Neural Information Processing Systems 34, NeurIPS*, 2021. (Cited on p. 114.)

John L. Kelly. A new interpretation of information rate. *The Bell System Technical Journal*, 1956. (Cited on p. 2.)

Hamza Keurti, Hsiao-Ru Pan, Michel Besserve, Benjamin F. Grewe, and Bernhard Schölkopf. Homomorphism autoencoder - learning group structured representations from observed transitions. In *Proceedings of the 39th International Conference on Machine Learning, ICML*, 2023. (Cited on p. 113.)

Mohammad Emtiyaz Khan, Didrik Nielsen, Voot Tangkaratt, Wu Lin, Yarin Gal, and Akash Srivastava. Fast and scalable Bayesian deep learning by weight-perturbation in Adam. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, 2018. (Cited on pp. 12, 50, and 56.)

Mohammad Emtiyaz Khan, Alexander Immer, Ehsan Abedi, and Maciej Korzepa. Approximate inference turns deep networks into Gaussian processes. In *Advances in Neural Information Processing Systems 32, NeurIPS*, 2019. (Cited on pp. 58 and 73.)

Jinwoo Kim, Dat Nguyen, Ayhan Suleymanzade, Hyeokjun An, and Seunghoon Hong. Learning probabilistic symmetrization for architecture agnostic equivariance. In *Advances in Neural Information Processing Systems 36, NeurIPS*, 2023. (Cited on p. 113.)

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR*, 2015. (Cited on p. 12.)

Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems 31, NeurIPS*, 2018. (Cited on p. 21.)

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. In *2nd International Conference on Learning Representations, ICLR*, 2014. (Cited on pp. 11 and 19.)

Diederik P. Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems 28, NeurIPS*, 2015. (Cited on pp. 12, 31, 50, 153, and 160.)

Diederik P. Kingma, Tim Salimans, and Max Welling. Improving variational inference with inverse autoregressive flows. *arXiv preprint: 1606.04934*, 2016. (Cited on pp. 21 and 23.)

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 2017. (Cited on p. 63.)

Andreas Kirsch, Sebastian Farquhar, Parmida Atighehchian, Andrew Jesson, Frédéric Branchaud-Charron, and Yarin Gal. Stochastic batch acquisition: A simple baseline for deep active learning. *Transactions on Machine Learning Research, TMLR*, 2023. (Cited on p. 48.)

Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (BiT): General visual representation learning. In *European Conference on Computer Vision, ECCV*, 2020. (Cited on p. 82.)

Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3D object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, 2013. (Cited on p. 171.)

Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. Being Bayesian, even just a bit, fixes overconfidence in ReLU networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, 2020. (Cited on pp. 74 and 166.)

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. (Cited on pp. 82, 161, 167, 171, 172, and 173.)

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25, NeurIPS*, 2012. (Cited on p. 1.)

Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation, and active learning. In *Advances in Neural Information Processing Systems 8, NeurIPS*, 1995. (Cited on pp. 8 and 81.)

Paul H. Kupiec. Techniques for verifying the accuracy of risk measurement models. *The Journal of Derivatives*, 1995. (Cited on p. 36.)

Anna Kuzina, Kumar Pratik, Fabio Valerio Massoli, and Arash Behboodi. Equivariant priors for compressed sensing with unknown orientation. In *Proceedings of the 38th International Conference on Machine Learning, ICML*, 2022. (Cited on p. 114.)

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems 30, NeurIPS*, 2017. (Cited on pp. 8, 15, 26, 31, 35, 50, 67, 71, 74, 81, 153, and 200.)

Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011. (Cited on p. 17.)

Neil D. Lawrence. *Variational inference in probabilistic models*. PhD thesis, University of Cambridge, 2001a. (Cited on p. 58.)

Neil D. Lawrence. Node relevance determination. In *Proceedings of the 12th Italian Workshop on Neural Nets*, 2001b. (Cited on pp. 14 and 50.)

Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1989. (Cited on pp. 99, 152, 161, and 167.)

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 2015. (Cited on p. 1.)

Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as Gaussian processes. In *6th International Conference on Learning Representations, ICLR*, 2018. (Cited on p. 12.)

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, 2019. (Cited on p. 99.)

Stefan Lee, Senthil Purushwalkam, Michael Cogswell, David Crandall, and Dhruv Batra. Why M heads are better than one: Training a diverse ensemble of deep networks. *arXiv preprint: 1511.06314*, 2015. (Cited on p. 95.)

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling giant models with conditional computation and automatic sharding. In *9th International Conference on Learning Representations, ICLR*, 2021. (Cited on pp. 78, 80, and 94.)

Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, 2011. (Cited on pp. 26 and 78.)

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes. In *6th International Conference on Learning Representations, ICLR*, 2018. (Cited on p. 13.)

Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research, JMLR*, 2017. (Cited on p. 153.)

Weitang Liu, Xiaoyun Wang, John D. Owens, and Yixuan Li. Energy-based out-of-distribution detection. In *Advances in Neural Information Processing Systems 33, NeurIPS*, 2020. (Cited on p. 18.)

Xiaolong Liu, Zhidong Deng, and Yuhan Yang. Recent progress in semantic image segmentation. *Artificial Intelligence Review*, 2019. (Cited on p. 73.)

Ekaterina Lobacheva, Nadezhda Chirkova, Maxim Kodryan, and Dmitry P. Vetrov. On power laws in deep ensembles. In *Advances in Neural Information Processing Systems 33, NeurIPS*, 2020. (Cited on p. 71.)

Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Raetsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, 2019. (Cited on p. 23.)

Raphael Gontijo Lopes, Yann N. Dauphin, and Ekin Dogus Cubuk. No one representation to rule them all: Overlapping features of training methods. In *10th International Conference on Learning Representations, ICLR*, 2022. (Cited on p. 94.)

Yuxuan Lou, Fuzhao Xue, Zangwei Zheng, and Yang You. Sparse-MLP: A fully-MLP architecture with conditional computation. *arXiv preprint: 2109.02008*, 2021. (Cited on p. 94.)

Christos Louizos and Max Welling. Structured and efficient variational deep learning with matrix Gaussian posteriors. In *Proceedings of the 32nd International Conference on Machine Learning, ICML*, 2016. (Cited on p. 74.)

Christos Louizos and Max Welling. Multiplicative normalizing flows for variational Bayesian neural networks. In *Proceedings of the 33rd International Conference on Machine Learning, ICML*, 2017. (Cited on p. 21.)

Christos Louizos, Kevin Swersky, Yujia Li, Max Welling, and Richard S. Zemel. The variational fair autoencoder. In *4th International Conference on Learning Representations, ICLR*, 2016. (Cited on p. 114.)

Chao Ma, Yingzhen Li, and José Miguel Hernández-Lobato. Variational implicit processes. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, 2019. (Cited on pp. 13 and 50.)

David J. C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 1992. (Cited on pp. 10, 50, 57, 73, 74, and 95.)

David J. C. MacKay. Bayesian nonlinear modeling for the prediction competition. *ASHRAE transactions*, 1994. (Cited on pp. 14 and 50.)

David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*, chapter 28. Model Comparison and Occam's Razor, pages 341–356. Cambridge University Press, 2003. (Cited on p. 2.)

Wesley J. Maddox, Pavel Izmailov, Timur Garipov, Dmitry P. Vetrov, and Andrew Gordon Wilson. A simple baseline for Bayesian uncertainty in deep learning. In *Advances in Neural Information Processing Systems 32, NeurIPS*, 2019. (Cited on pp. 8, 50, 63, 67, and 74.)

Wesley J. Maddox, Gregory W. Benton, and Andrew Gordon Wilson. Rethinking parameter counting in deep models: Effective dimensionality revisited. *arXiv preprint 2003.02139*, 2020. (Cited on pp. 56 and 66.)

Kaitlin Maile, Dennis George Wilson, and Patrick Forré. Equivariance-aware architectural optimization of neural networks. In *11th International Conference on Learning Representations, ICLR*, 2023. (Cited on p. 104.)

Xiaofeng Mao, Gege Qi, Yuefeng Chen, Xiaodan Li, Ranjie Duan, Shaokai Ye, Yuan He, and Hui Xue. Towards robust vision transformer. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2022. (Cited on p. 94.)

James Martens. *Second-order Optimization for Neural Networks*. PhD thesis, University of Toronto, 2016. (Cited on p. 58.)

James Martens. New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research, JMLR*, 2020. (Cited on p. 58.)

James Martens and Roger B. Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *Proceedings of the 39th International Conference on Machine Learning, ICML*, 2015. (Cited on pp. 10 and 120.)

James Martens and Ilya Sutskever. Learning recurrent neural networks with Hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning, ICML*, 2011. (Cited on p. 58.)

Loic Matthey, Irina Higgins, Demis Hassabis, and Alexander Lerchner. dsprites: Disentanglement testing sprites dataset, 2017. (Cited on pp. 108 and 218.)

Ning Miao, Tom Rainforth, Emile Mathieu, Yann Dubois, Yee Whye Teh, Adam Foster, and Hyunjik Kim. Learning instance-specific augmentations by capturing local invariances. In *Proceedings of the 39th International Conference on Machine Learning, ICML*, 2023. (Cited on pp. 99, 113, and 121.)

Xu Miao and Rajesh P. N. Rao. Learning the lie groups of visual invariance. *Neural Computation*, 19(10):2665–2693, 2007. (Cited on p. 113.)

Matthias Minderer, Josip Djolonga, Rob Romijnders, Frances Hubis, Xiaohua Zhai, Neil Houlsby, Dustin Tran, and Mario Lucic. Revisiting the calibration of modern neural networks. In *Advances in Neural Information Processing Systems 34, NeurIPS*, 2021. (Cited on pp. 92 and 94.)

Thomas P Minka. Bayesian model averaging is not model combination. *Available electronically at http://www. stat. cmu. edu/minka/papers/bma. html*, 2000. (Cited on p. 50.)

Aaron Mishkin, Frederik Kunstner, Didrik Nielsen, Mark Schmidt, and Mohammad Emtiyaz Khan. SLANG: fast structured covariance approximations for Bayesian deep learning with natural gradient. In *Advances in Neural Information Processing Systems 31, NeurIPS*, 2018. (Cited on p. 73.)

Bruno Kacper Mlodozeniec, Matthias Reisser, and Christos Louizos. Hyperparameter optimization through neural network partitioning. In *11th International Conference on Learning Representations, ICLR*, 2023. (Cited on p. 99.)

Arnab Kumar Mondal, Siba Smarak Panigrahi, Oumar Kaba, Sai Mudumba, and Siamak Ravanbakhsh. Equivariant adaptation of large pretrained models. In *Advances in Neural Information Processing Systems 36, NeurIPS*, 2023. (Cited on p. 113.)

Chelsea Murray, James U Allingham, Javier Antorán, and José Miguel Hernández-Lobato. Addressing bias in active learning with depth uncertainty networks... or not. In *I (Still) Can't Believe It's Not Better! Workshop at NeurIPS 2021*, 2021a. (Cited on p. 121.)

Chelsea Murray, James Urquhart Allingham, Javier Antorán, and José Miguel Hernández-Lobato. Depth uncertainty networks for active learning. *NeurIPS Workshop on Bayesian Deep Learning*, 2021b. (Cited on pp. 3, 25, and 121.)

Basil Mustafa, Carlos Riquelme, Joan Puigcerver, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Deep ensembles for low-data transfer learning. *arXiv preprint: 2010.06866*, 2020. (Cited on pp. 90 and 201.)

Eric T. Nalisnick and Padhraic Smyth. Learning approximately objective priors. In *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, August 11-15, 2017*, 2017. (Cited on p. 114.)

Eric T. Nalisnick and Padhraic Smyth. Learning priors for invariance. In *The 21st International Conference on Artificial Intelligence and Statistics, AISTATS*, 2018. (Cited on pp. 13, 61, and 99.)

Eric T. Nalisnick, José Miguel Hernández-Lobato, and Padhraic Smyth. Dropout as a structured shrinkage prior. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, 2019a. (Cited on pp. 14 and 50.)

Eric T. Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Görür, and Balaji Lakshminarayanan. Do deep generative models know what they don't know? In *7th International Conference on Learning Representations, ICLR*, 2019b. (Cited on p. 72.)

Eric T. Nalisnick, Jonathan Gordon, and José Miguel Hernández-Lobato. Predictive complexity priors. In *The 24th International Conference on Artificial Intelligence and Statistics, AISTATS*, 2021. (Cited on pp. 13 and 61.)

Giung Nam, Jongmin Yoon, Yoonho Lee, and Juho Lee. Diversity matters when learning from ensembles. In *Advances in Neural Information Processing Systems 34, NeurIPS*, 2021. (Cited on p. 95.)

Radford M. Neal. *Bayesian learning for neural networks*. PhD thesis, University of Toronto, 1995. (Cited on pp. 11, 12, 49, and 73.)

Radford M. Neal. MCMC using Hamiltonian dynamics. *Handbook of Markov chain Monte Carlo*, 2011. (Cited on p. 11.)

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *Advances in Neural Information Processing Systems 24, NeurIPS*, 2011. (Cited on pp. 161, 167, and 173.)

Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2015. (Cited on pp. 26 and 54.)

Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, 2008. (Cited on p. 173.)

Jeremy Nixon, Michael W. Dusenberry, Linchuan Zhang, Ghassen Jerfel, and Dustin Tran. Measuring calibration in deep learning. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops*, 2019. (Cited on p. 17.)

Sebastian W. Ober and Carl Edward Rasmussen. Benchmarking the neural linear model for regression. In *1st Symposium on Advances in Approximate Bayesian Inference, AABI*, 2019. (Cited on p. 74.)

OpenAI. GPT-4 technical report. *arXiv preprint: 2303.08774*, 2023. (Cited on p. 1.)

David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of artificial intelligence research*, 1999. (Cited on pp. 8 and 81.)

Kazuki Osawa, Siddharth Swaroop, Mohammad Emtiyaz Khan, Anirudh Jain, Runa Eschenhagen, Richard E. Turner, and Rio Yokota. Practical deep learning with Bayesian principles. In *Advances in Neural Information Processing Systems 32, NeurIPS*, 2019. (Cited on pp. 12, 50, 54, 56, 63, 67, 74, and 166.)

Ian Osband. Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout. In *NeurIPS workshop on Bayesian deep learning*, 2016. (Cited on p. 12.)

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. In *Advances in Neural Information Processing Systems 29, NeurIPS*, 2016. (Cited on p. 2.)

Yaniv Ovadia, Emily Fertig, Balaji Lakshminarayanan, Sebastian Nowozin, D. Sculley, Joshua V. Dillon, Jie Ren, Zachary Nado, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *Advances in Neural Information Processing Systems 32, NeurIPS*, 2019. (Cited on pp. 17, 31, 40, 41, 54, 63, 67, 69, 74, 79, 158, and 200.)

George Papamakarios, Iain Murray, and Theo Pavlakou. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems 30, NeurIPS*, 2017. (Cited on p. 21.)

George Papamakarios, Eric T. Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research, JMLR*, 2021. (Cited on p. 21.)

Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and CV Jawahar. Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2012. (Cited on pp. 171 and 173.)

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32, NeurIPS*, 2019. (Cited on p. 152.)

David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. *arXiv preprint: 2104.10350*, 2021. (Cited on pp. 78 and 120.)

Sayak Paul and Pin-Yu Chen. Vision transformers are robust learners. In *The 36th AAAI Conference on Artificial Intelligence, AAAI*, 2022. (Cited on p. 94.)

Tim Pearce, Russell Tsuchida, Mohamed Zaki, Alexandra Brintrup, and Andy Neely. Expressive priors in Bayesian neural networks: Kernel combinations and periodic functions. In *Proceedings of the 35th Conference on Uncertainty in Artificial Intelligence, UAI*, 2019. (Cited on pp. 13 and 61.)

Robert Pinsler, Jonathan Gordon, Eric T. Nalisnick, and José Miguel Hernández-Lobato. Bayesian batch active learning as sparse subset approximation. In *Advances in Neural Information Processing Systems 32, NeurIPS*, 2019. (Cited on p. 74.)

Janis Postels, Francesco Ferroni, Huseyin Coskun, Nassir Navab, and Federico Tombari. Sampling-free epistemic uncertainty estimation using approximated variance propagation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision, ICCV*, 2019. (Cited on p. 50.)

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *Proceedings of the 37th International Conference on Machine Learning, ICML*, 2021. (Cited on p. 82.)

Alexandre Ramé, Rémy Sun, and Matthieu Cord. Mixmo: Mixing multiple inputs for multiple outputs via deep subnetworks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision, ICCV*, 2021. (Cited on p. 95.)

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with CLIP latents. *arXiv preprint: 2204.06125*, 2022. (Cited on pp. 1 and 19.)

Rajesh Ranganath, Sean Gerrish, and David M. Blei. Black box variational inference. In *The 7th International Conference on Artificial Intelligence and Statistics, AISTATS*, 2014. (Cited on p. 10.)

Rajesh P. N. Rao and Daniel L. Ruderman. Learning lie groups for invariant visual perception. In Michael J. Kearns, Sara A. Solla, and David A. Cohn, editors, *Advances in Neural Information Processing Systems 11, NeurIPS*, 1998. (Cited on p. 113.)

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do ImageNet classifiers generalize to ImageNet? In *Proceedings of the 35th International Conference on Machine Learning, ICML*, 2019. (Cited on p. 172.)

Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the 31st International Conference on Machine Learning, ICML*, 2015. (Cited on p. 21.)

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 30th International Conference on Machine Learning, ICML*, 2014. (Cited on p. 19.)

Oren Rippel and Ryan Prescott Adams. High-dimensional probability estimation with deep density models. *arXiv preprint: 1302.5125*, 2013. (Cited on p. 19.)

Carlos Riquelme, George Tucker, and Jasper Snoek. Deep Bayesian bandits showdown: An empirical comparison of Bayesian deep networks for thompson sampling. In *6th International Conference on Learning Representations, ICLR*, 2018. (Cited on p. 74.)

Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts. In *Advances in Neural Information Processing Systems 34, NeurIPS*, 2021. (Cited on pp. 78, 79, 80, 81, 82, 83, 84, 88, 90, 94, 169, 170, 171, 174, 175, 176, 181, 182, 190, and 200.)

Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable Laplace approximation for neural networks. In *6th International Conference on Learning Representations, ICLR*, 2018. (Cited on pp. 10, 50, 59, 74, 95, 120, and 166.)

David W. Romero and Suhas Lohit. Learning partial equivariances from data. In *Advances in Neural Information Processing Systems 35, NeurIPS*, 2022. (Cited on p. 99.)

Cédric Rommel, Thomas Moreau, and Alexandre Gramfort. Deep invariant networks with differentiable augmentation layers. In *Advances in Neural Information Processing Systems 35, NeurIPS*, 2022. (Cited on p. 99.)

Simone Rossi, Sébastien Marmin, and Maurizio Filippone. Walsh-hadamard variational inference for Bayesian deep learning. In *Advances in Neural Information Processing Systems 33, NeurIPS*, 2020. (Cited on p. 74.)

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 2015. (Cited on pp. 1 and 43.)

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L. Denton, Seyed Kamyar Seyed Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. In *Advances in Neural Information Processing Systems 35, NeurIPS*, 2022. (Cited on pp. 1 and 19.)

Ruslan Salakhutdinov and Geoffrey Hinton. Deep boltzmann machines. In *The 12th International Conference on Artificial Intelligence and Statistics, AISTATS*, 2009. (Cited on p. 17.)

Pola Schwöbel, Martin Jørgensen, Sebastian W. Ober, and Mark van der Wilk. Last layer marginal likelihood for invariance learning. In *The 25th International Conference on Artificial Intelligence and Statistics, AISTATS*, 2022. (Cited on p. 99.)

Andrew W. Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Zídek, Alexander W. R. Nelson, Alex Bridgland, Hugo Penedones, Stig Petersen, Karen Simonyan, Steve Crossan, Pushmeet Kohli, David T. Jones, David Silver, Koray Kavukcuoglu, and Demis Hassabis. Improved protein structure prediction using potentials from deep learning. *Nature*, 2020. (Cited on p. 1.)

Burr Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009. (Cited on pp. 2 and 48.)

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *5th International Conference on Learning Representations, ICLR*, 2017. (Cited on pp. 3, 78, 80, 81, 94, and 175.)

Zhixin Shu, Mihir Sahasrabudhe, Riza Alp Güler, Dimitris Samaras, Nikos Paragios, and Iasonas Kokkinos. Deforming autoencoders: Unsupervised disentangling of shape and appearance. In *European Conference on Computer Vision, ECCV*, 2018. (Cited on p. 114.)

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 2016. (Cited on p. 1.)

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR*, 2015. (Cited on p. 152.)

Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, Colorado Univ at Boulder Dept of Computer Science, 1986. (Cited on p. 17.)

Edward Lloyd Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems 18, NeurIPS*, 2005. (Cited on p. 12.)

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25, NeurIPS*, 2012. (Cited on p. 153.)

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Prabhat, and Ryan P. Adams. Scalable Bayesian optimization using deep neural networks. In *Proceedings of the 31st International Conference on Machine Learning, ICML*, 2015. (Cited on pp. 56, 64, and 74.)

Masoumeh Soflaei, Hongyu Guo, Ali Al-Bashabsheh, Yongyi Mao, and Richong Zhang. Aggregated learning: A vector-quantization approach to learning neural network classifiers. In *The 34th AAAI Conference on Artificial Intelligence, AAAI*, 2020. (Cited on p. 89.)

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proceedings of the 31st International Conference on Machine Learning, ICML*, 2015. (Cited on p. 18.)

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *9th International Conference on Learning Representations, ICLR*, 2021. (Cited on p. 18.)

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research, JMLR*, 2014. (Cited on pp. 12 and 200.)

Andreas Steiner, Alexander Kolesnikov, Xiaohua Zhai, Ross Wightman, Jakob Uszkoreit, and Lucas Beyer. How to train your ViT? data, augmentation, and regularization in vision transformers. *Transactions on Machine Learning Research, TMLR*, 2022. (Cited on p. 200.)

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL*, 2019. (Cited on p. 78.)

Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE/CVF International Conference on Computer Vision, ICCV*, 2017. (Cited on pp. 82 and 169.)

Shengyang Sun, Guodong Zhang, Jiaxin Shi, and Roger B. Grosse. Functional variational Bayesian neural networks. In *7th International Conference on Learning Representations, ICLR*, 2019. (Cited on pp. 13, 50, and 61.)

Jakub Swiatkowski, Kevin Roth, Bastiaan S. Veeling, Linh Tran, Joshua V. Dillon, Jasper Snoek, Stephan Mandt, Tim Salimans, Rodolphe Jenatton, and Sebastian Nowozin. The k-tied normal distribution: A compact parameterization of Gaussian mean field posteriors in Bayesian neural networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, 2020. (Cited on p. 74.)

Tesla. Autopilot | Tesla. https://www.tesla.com/autopilot, 2024. Accessed: 2024-02-05. (Cited on p. 2.)

Mattias Teye, Hossein Azizpour, and Kevin Smith. Bayesian uncertainty estimation for batch normalized deep networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, 2018. (Cited on pp. 12 and 50.)

Michalis K. Titsias. Variational learning of inducing variables in sparse gaussian processes. In *The 12th International Conference on Artificial Intelligence and Statistics, AISTATS*, 2009. (Cited on p. 12.)

Linh Tran, Bastiaan S. Veeling, Kevin Roth, Jakub Swiatkowski, Joshua V. Dillon, Jasper Snoek, Stephan Mandt, Tim Salimans, Sebastian Nowozin, and Rodolphe Jenatton. Hydra: Preserving ensemble diversity for model distillation. *arXiv preprint: 2001.04694*, 2020. (Cited on p. 95.)

Benigno Uria, Iain Murray, and Hugo Larochelle. RNADE: the real-valued neural autoregressive density-estimator. In *Advances in Neural Information Processing Systems 26, NeurIPS*, 2013. (Cited on p. 17.)

Sharvaree Vadgama, Jakub Mikolaj Tomczak, and Erik J Bekkers. Kendall shape-VAE: Learning shapes in a generative framework. In *NeurIPS 2022 Workshop on Symmetry and Geometry in Neural Representations*, 2022. (Cited on p. 114.)

Arash Vahdat and Jan Kautz. NVAE: A deep hierarchical variational autoencoder. In *Advances in Neural Information Processing Systems 33, NeurIPS*, 2020. (Cited on p. 23.)

Rianne van den Berg, Leonard Hasenclever, Jakub M. Tomczak, and Max Welling. Sylvester normalizing flows for variational inference. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, 2018. (Cited on p. 21.)

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. WaveNet: A generative model for raw audio. In *The 9th ISCA Speech Synthesis Workshop, Sunnyvale, CA, USA, 13-15 September 2016*, 2016a. (Cited on p. 17.)

Aäron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Koray Kavukcuoglu, Oriol Vinyals, and Alex Graves. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems 29, NeurIPS*, 2016b. (Cited on p. 17.)

Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *Proceedings of the 32nd International Conference on Machine Learning, ICML*, 2016c. (Cited on p. 17.)

Aaron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George Driessche, Edward Lockhart, Luis Cobo, Florian Stimberg, Norman Casagrande, Dominik Grewe, Seb Noury, Sander Dieleman, Erich Elsen, Nal Kalchbrenner, Heiga Zen, Alex Graves, Helen King, Tom Walters, Dan Belov, and Demis Hassabis. Parallel WaveNet: Fast high-fidelity speech synthesis. In *International Conference on Machine Learning*, 2018. (Cited on p. 21.)

Tycho F. A. van der Ouderaa and Mark van der Wilk. Learning invariant weights in neural networks. In *Proceedings of the 38th Conference on Uncertainty in Artificial Intelligence, UAI*, 2022. (Cited on pp. 99, 102, 106, and 211.)

Mark van der Wilk, Matthias Bauer, S. T. John, and James Hensman. Learning invariances using the marginal likelihood. In *Advances in Neural Information Processing Systems 31, NeurIPS*, 2018. (Cited on pp. 99 and 106.)

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30, NeurIPS*, 2017. (Cited on pp. 3 and 80.)

Bastiaan S. Veeling, Jasper Linmans, Jim Winkens, Taco Cohen, and Max Welling. Rotation equivariant cnns for digital pathology. *arXiv preprint: 1806.03962*, 2018. (Cited on pp. 108 and 224.)

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Çaglar Gülçehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 2019. (Cited on p. 1.)

Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The Caltech-UCSD birds-200-2011 dataset. Technical report, California Institute of Technology, 2011. (Cited on p. 171.)

Mike Walmsley, Chris Lintott, Tobias Géron, Sandor Kruk, Coleman Krawczyk, Kyle W. Willett, Steven Bamford, Lee S. Kelvin, Lucy Fortson, Yarin Gal, William Keel, Karen L. Masters, Vihang Mehta, Brooke D. Simmons, Rebecca Smethurst, Lewis Smith, Elisabeth M. Baeten, and Christine Macmillan. Galaxy Zoo DECaLS: Detailed visual morphology measurements from volunteers and deep learning for 314 000 galaxies. *arXiv preprint: 2102.08414*, 2022. (Cited on p. 108.)

Chaoqi Wang, Guodong Zhang, and Roger B. Grosse. Picking winning tickets before training by preserving gradient flow. In *8th International Conference on Learning Representations, ICLR*, 2020. (Cited on p. 74.)

Ziyu Wang, Tongzheng Ren, Jun Zhu, and Bo Zhang. Function space particle optimization for Bayesian neural networks. In *7th International Conference on Learning Representations, ICLR*, 2019. (Cited on p. 50.)

Max Welling. Do we still need models or just more data and compute? *University of Amsterdam, April*, 2019. (Cited on p. 19.)

Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 27th International Conference on Machine Learning, ICML*, 2011. (Cited on p. 11.)

Yeming Wen, Dustin Tran, and Jimmy Ba. BatchEnsemble: an alternative approach to efficient ensemble and lifelong learning. In *8th International Conference on Learning Representations, ICLR*, 2020. (Cited on pp. 8, 13, 81, 82, 85, 89, 95, 174, 175, 179, 180, and 182.)

Florian Wenzel, Kevin Roth, Bastiaan S. Veeling, Jakub Swiatkowski, Linh Tran, Stephan Mandt, Jasper Snoek, Tim Salimans, Rodolphe Jenatton, and Sebastian Nowozin. How good is the Bayes posterior in deep neural networks really? In *Proceedings of the 36th International Conference on Machine Learning, ICML*, 2020a. (Cited on pp. 13 and 50.)

Florian Wenzel, Jasper Snoek, Dustin Tran, and Rodolphe Jenatton. Hyperparameter ensembles for robustness and uncertainty quantification. In *Advances in Neural Information Processing Systems 33, NeurIPS*, 2020b. (Cited on pp. 94, 95, and 175.)

Andrew Gordon Wilson. The case for Bayesian deep learning. *arXiv preprint 2001.10995*, 2020. (Cited on pp. 13, 50, and 77.)

Andrew Gordon Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. In *Advances in Neural Information Processing Systems 33, NeurIPS*, 2020. (Cited on pp. 13, 74, and 77.)

Robin Winter, Marco Bertolini, Tuan Le, Frank Noé, and Djork-Arné Clevert. Unsupervised learning of group invariant and equivariant representations. In *Advances in Neural Information Processing Systems 35, NeurIPS*, 2022. (Cited on p. 113.)

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato,

Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint: 1609.08144*, 2016. (Cited on p. 1.)

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint: 1708.07747*, 2017. (Cited on pp. 161 and 167.)

Jingjing Xie, Bing Xu, and Chuang Zhang. Horizontal and vertical ensemble with deep representation for classification. *arXiv preprint: 1306.2759*, 2013. (Cited on p. 95.)

Jin Xu, Hyunjik Kim, Thomas Rainforth, and Yee Whye Teh. Group equivariant subsampling. In *Advances in Neural Information Processing Systems 34, NeurIPS*, 2021. (Cited on p. 114.)

Fuzhao Xue, Ziji Shi, Futao Wei, Yuxuan Lou, Yong Liu, and Yang You. Go wider instead of deeper. In *The 36th AAAI Conference on Artificial Intelligence, AAAI*, 2022. (Cited on p. 94.)

An Yang, Junyang Lin, Rui Men, Chang Zhou, Le Jiang, Xianyan Jia, Ang Wang, Jie Zhang, Jiamang Wang, Yong Li, et al. Exploring sparse expert models and beyond. *arXiv preprint: 2105.15082*, 2021. (Cited on pp. 85, 87, and 94.)

Jianke Yang, Robin Walters, Nima Dehmamy, and Rose Yu. Generative adversarial symmetry discovery. In *Proceedings of the 39th International Conference on Machine Learning, ICML*, 2023. (Cited on pp. 113, 223, and 224.)

Yi Yang and Shawn Newsam. Bag-of-visual-words and spatial extensions for land-use classification. In *Proceedings of the 18th SIGSPATIAL international conference on advances in geographic information systems*, 2010. (Cited on p. 172.)

Seniha Esen Yuksel, Joseph N Wilson, and Paul D Gader. Twenty years of mixture of experts. *IEEE transactions on neural networks and learning systems*, 2012. (Cited on p. 94.)

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*, 2016. (Cited on pp. 157 and 165.)

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision, ECCV*, 2014. (Cited on p. 28.)

Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2022. (Cited on p. 183.)

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *6th International Conference on Learning Representations, ICLR*, 2018. (Cited on p. 200.)

Ruqi Zhang, Chunyuan Li, Jianyi Zhang, Changyou Chen, and Andrew Gordon Wilson. Cyclical stochastic gradient MCMC for Bayesian deep learning. In *7th International Conference on Learning Representations, ICLR*, 2019. (Cited on p. 95.)

Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. (Cited on p. 173.)

Appendix A

DUN Details

A.1 Shape and Size Adaptation

When applying Depth Uncertainty Networks (DUNs) to modern CNN architectures, such as ResNets, there is an additional complication. The output block of a DUN expects a certain number of channels in its input. However, the intermediate blocks have inconsistent numbers of channels in their outputs. To overcome this issue, we introduce *adaptation layers* which up-scale the number of channels of intermediate layers so that they match the number expected by the output block. This is shown in Figure A.1.

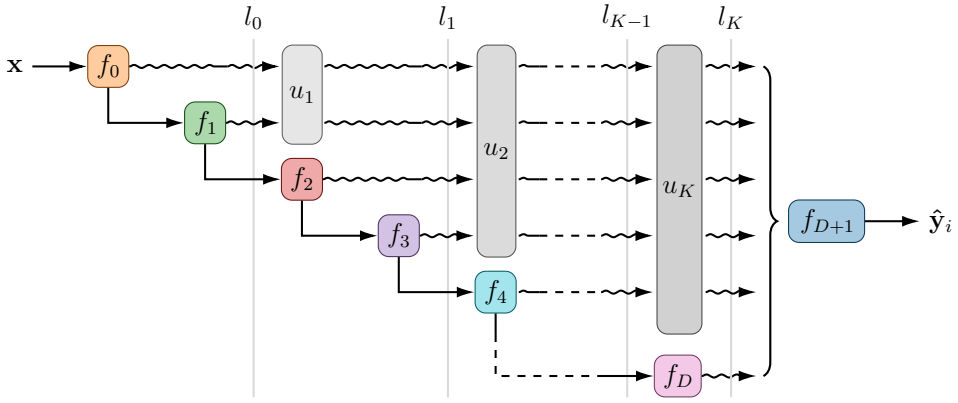


Figure A.1: For network architectures in which the input and output number of channels or dimensions is not constant, we add *adaptation layers* to the computational model shown in Figure 3.2. The n^{th} adaption layer u_n takes a number of channels/dimensions l_{n-1} and outputs l_n channels/dimensions. Later adaptation layers are reused multiple times, reducing the number of parameters required. Note that block sizes are unrelated to their number of parameters.

Note, however, that this channel mismatch issue is a specific instance of a more general problem of shape and size mismatch between layers in a DUN. Consider the following cases where constructing a DUN is non-trivial:

- A NN consisting of series of dense layers of different dimensions. E.g., an auto-encoder or U-Net.
- A NN consisting of a mix of convolutional and dense layers. E.g., VGG (Simonyan and Zisserman, 2015) or LeNet (LeCun et al., 1989).

In the first case, we will have dimensionality mismatches between the different dense layers. In the second case, we will have shape mismatches between the 3D convolutional layers and the 1D dense layers, in addition to the potential size mismatches between layers if the same type.

Fortunately, adaptation layers can be used to solve any shape and size mismatches. Size mismatches can be naively solved by either padding or dropping tensors as appropriate. Another solution is to use (parameter) cheap 1×1 convolution layers and low-rank[1] dense layers in the case of mismatches between the number of channels and the number of dimensions, respectively. Shape mismatches can easily be solved by using standard reshape and/or flatten layers found in most deep learning frameworks.

A.2 Experimental Setup

We implement all of our experiments in PyTorch (Paszke et al., 2019). Gaussian processes for toy data experiments are implemented with GPyTorch (Gardner et al., 2018).

A.2.1 Toy Dataset Experiments

Unless specified otherwise, all NNs used for toy regression experiments in Section 3.3.2 consist of fully connected models with ReLU activations and residual connections. Their hidden layer width is 100. Batch normalisation is applied after every layer for Stochastic Gradient Descent (SGD) and DUNs. Network depths are defined on a per-experiment basis. DUNs employ linear input and output blocks, meaning that a depth of d=0 corresponds to a linear model. We refer to depth as the number of hidden layers of a NN.

Ensemble elements, DUNs and dropout models employ a weight decay value of 10^{-4}. Ensembles are composed of 20 identical networks, trained from different initialisations.

[1] A low rank dense layer with input of size n_1 and output of size n_2 can be constructed by composing two standard dense layers of size $n_1\times n_3$ and $n_3\times n_2$ where $n_3 << n_1, n_2$.

Initialisation parameters are sampled from the He initialisation (He et al., 2015). Dropout probabilities are fixed to 0.1. Mean-Field Variational Inference (MFVI) networks use a $\mathcal{N}(\mathbf{0}, I)$ prior. Gradients of the likelihood term in the Evidence Lower BOund (ELBO) are estimated with the local reparameterisation trick (Kingma et al., 2015) using 5 Monte Carlo (MC) samples. DUNs employ uniform priors, assigning the same mass to each depth.

Networks are optimised using 6000 steps of full-batch gradient descent with a momentum value of 0.9 and learning rate of 10^{-3}. Exceptions to this are: Dropout being trained for 10000 epochs, as we found 6000 to not be enough to achieve convergence, and MFVI using a learning rate of 10^{-2}. For MFVI and DUNs, we scale the ELBO by one over the number of data points N. This makes the scale of the objective insensitive to dataset size.

The parameters of the predictive distributions are computed as described in Section 2.1.3. For 1D datasets, we draw 10^4 MC samples with MFVI and dropout. Plot error bars correspond to the standard deviations of each approach's mean predictions. Thus, they convey model uncertainty.

Gaussian processes use a Gaussian likelihood function and radial basis function kernel. A gamma prior with parameters $\alpha = 1, \beta = 20$ is placed on the length-scale parameter. This avoids local optima of the log-likelihood function where fast-varying patterns in the data are treated like noise. Noise variance and kernel parameters are learnt by optimising the Marginal Log Likelihood (MLL) with 100 steps of Adam. Step size is set to 0.1.

We employ 7 different toy datasets. These allow us to test methods' capacity to express uncertainty in-between clusters of data and outside the convex hull of the data. They also allow us to evaluate methods' capacity to fit different, quickly varying functions. All of them can be loaded using our provided code.

A.2.2 Regression Experiments

Hyperparameter Optimisation and Training

To obtain our results on tabular regression tasks, given in Section 3.3.3, we follow Hernández-Lobato and Adams (2015) and follow-up work Gal and Ghahramani (2016); Lakshminarayanan et al. (2017) in performing Hyperparameter Optimisation (HPO) to determine the best configurations for each method. However, rather than using Bayesian Optimisation (BO) (Snoek et al., 2012) we use Bayesian Optimisation and Hyperband (BOHB) (Falkner et al., 2018). This method, as the name suggests, combines BO with Hyperband, a bandit-based HPO method (Li et al., 2017). BOHB has the

strengths of both BO (strong final performance) and Hyperband (scalability, flexibility, and robustness).

In particular, we use the `HpBandSter` implementation of BOHB: https://github.com/automl/HpBandSter. We run BOHB for each dataset and split for 20 iterations using the same settings, shown in Table A.1. `min_budget` and `max_budget` are defined on a per-dataset basis, as shown in Table A.2. We find these values to be sufficiently large to ensure all methods' convergence.

Table A.1: BOHB settings.

Setting	Value
`eta`	3
`min_points_in_model`	None
`top_n_percent`	14
`num_samples`	64
`random_fraction`	1/3
`bandwidth_factor`	3
`min_bandwidth`	1e-3

For each test-train split of each dataset, we split the original training set into a new training set and a validation set. The validation sets are taken to be the last N elements of the original training set, where N is calculated from the validation proportions listed in Table A.2. The training and validation sets are normalised by subtracting the mean and dividing by the variance of the new training set. BOHB performs minimisation on the validation Negative Log Likelihood (NLL). During optimisation, we perform early stopping with patience values shown in Table A.2.

As shown in Table A.3, each method has a different set of hyperparameters to optimise. The BOHB configuration for each hyperparameter is shown in Table A.4. It is worth noting that maximum network depth is a hyperparameter which we optimise with BOHB. DUNs benefit from being deeper as it allows them to perform Bayesian Model Averaging (BMA) over a larger set of functions. We prevent this from disadvantaging competing methods by choosing the depth at which each one performs best.

All methods are applied to fully-connected networks with a hidden layer width of 100. We employ residual connections, allowing all approaches to better take advantage of depth. All methods are trained using SGD with momentum and a batch size of 128. No learning rate scheduling is performed. We use batch-normalisation for DUNs and vanilla networks (labelled SGD in experiments). All DUNs are trained using Variational Inference (VI) (3.7). The likelihood term in the MFVI ELBO is estimated with 3 MC

Table A.2: Per-dataset HPO configurations.

Dataset	Min Budget	Max Budget	Early Stop Patience	Val Prop
Boston	200	2000	200	0.15
Concrete	200	2000	200	0.15
Energy	200	2000	200	0.15
Kin8nm	50	500	50	0.15
Naval	50	500	50	0.15
Power	50	500	50	0.15
Protein	50	500	50	0.15
Wine	100	1000	100	0.15
Yacht	200	2000	200	0.15
Boston Gap	200	2000	200	0.15
Concrete Gap	200	2000	200	0.15
Energy Gap	200	2000	200	0.15
Kin8nm Gap	50	500	50	0.15
Naval Gap	50	500	50	0.15
Power Gap	50	500	50	0.15
Protein Gap	50	500	50	0.15
Wine Gap	100	1000	100	0.15
Yacht Gap	200	2000	200	0.15
Flights	2	25	5	0.05

samples per input. For MFVI and Dropout, 10 MC samples are used to estimate the test log-likelihood. Ensembles use 5 elements for prediction. Ensemble elements differ from each other in their initialisation, which is sampled from the He initialisation distribution (He et al., 2015). We do not use adversarial training as, in line with Ashukha et al. (2020), we do not find it to improve results.

Table A.3: Hyperparameters optimised for each method.

Hyperparameter	DUN	SGD	MFVI	MC Dropout
Learning Rate	✓	✓	✓	✓
SGD Momentum	✓	✓	✓	✓
Num. Layers	✓	✓	✓	✓
Weight Decay	✓	✓		✓
Prior Std. Dev.			✓	
Drop Prob.				✓

Table A.4: BOHB hyperparameter optimisation configurations. All hyperparameters were sampled from uniform distributions.

Hyperparameter	Lower	Upper	Default	Log	Data Type
Learning Rate	1×10^{-4}	1	0.01	`True`	`float`
SGD Momentum	0	0.99	0.5	`False`	`float`
Num. Layers	1	40	5	`False`	`int`
Weight Decay	1×10^{-6}	0.1	5×10^{-4}	`True`	`float`
Prior Std. Dev.	0.01	10	1	`True`	`float`
Drop Prob.	5×10^{-3}	0.5	0.2	`True`	`float`

Evaluation

The best configuration found for each dataset, method and split is used to re-train a model on the entire original training set. For the flights dataset, which does not come with multiple splits, we repeat this five times. We report mean and standard deviation values across all five. Final run training and test sets are normalised using the mean and variance of the original training set. Note, however, that the results presented in Section 3.3.3 are unnormalised. The number of epochs used for final training runs is the number of epochs at which the optimal configuration was found with HPO.

Timing experiments for regression models are performed on a 40-core Intel Xeon CPU E5-2650 v3 2.30GHz. We report computation time for a single batch of size 512,

which we evaluate across 5 runs. Ensembles, Dropout and MFVI require multiple forward passes per batch. We report the time taken for all passes to be made. For Ensembles, we also include network loading time.

A.2.3 Image Experiments

Training

The results shown in Section 3.3.4 are obtained by training ResNet-50 models using SGD with momentum. The initial learning rate, momentum, and weight decay are 0.1, 0.9, and 1×10^{-4}, respectively. We train on 2 Nvidia P100 GPUs with a batch size of 256 for all experiments. Each dataset is trained for a different number of epochs, shown in Table A.5. We decay the learning rate by a factor of 10 at scheduled epochs, also shown in Table A.5. Otherwise, all methods and datasets share hyperparameters. These hyperparameter settings are the defaults provided by `PyTorch` for training on ImageNet. We found them to perform well across the board. We report results obtained at the final training epoch. We do not use a separate validation set to determine the best epoch, as we found ResNet-50 not to overfit with the chosen schedules.

Table A.5: Per-dataset training configuration for image experiments.

Dataset	Num. Epochs	LR Schedule
MNIST	90	40, 70
Fashion	90	40, 70
SVHN	90	50, 70
CIFAR10	300	150, 225
CIFAR100	300	150, 225
ImageNet	90	30, 60

For dropout experiments, we add dropout to the standard ResNet-50 model (He et al., 2016a) in between the 2^{nd} and 3^{rd} convolutions in the bottleneck blocks. This approach follows Zagoruyko and Komodakis (2016) and Ashukha et al. (2020), who add dropout in between the two convolutions of a WideResNet-50's basic block. Following their approach, we try a dropout probability of 0.3. However, we find that this value is too large and causes underfitting. A dropout probability of 0.1 provides stronger results. We use 10 MC samples for predictions. Ensembles use 5 elements for prediction. Ensemble elements differ from each other in their initialisation, which is sampled from the He initialisation distribution (He et al., 2015). We do not use adversarial training as, in line with Ashukha et al. (2020), we do not find it to improve results.

We modify the standard ResNet-50 architecture such that the first 7×7 convolution is replaced with a 3×3 convolution. Additionally, we remove the first max-pooling layer. Following Goyal et al. (2017), we zero-initialise the last batch normalisation layer in residual blocks so that they act as identity functions at the start of training. Because the output block of a ResNet expects to receive activations with a fixed number of channels, we add *up-scaling layers*. We implement these using 1×1 convolutions. Figure A.1 shows this modified computational model.

For the MNIST and Fashion-MNIST datasets, we train DUNs with a fixed approximate posterior $q_{\boldsymbol{\alpha}}(d) = p_{\boldsymbol{\beta}}(d)$ for the first 3 epochs. These are the simplest image datasets we work with and can be readily solved with shallower models than ResNet-50. By fixing $q_{\boldsymbol{\alpha}}(d)$ for the first epochs, we ensure all layers receive strong gradients and become useful for making predictions.

Evaluation

All methods are trained 5 times on each dataset, allowing for error bars in experiments. We report mean values and standard deviations.

To evaluate the methods' resilience to out-of-distribution data, we follow Ovadia et al. (2019). We train each method on MNIST and evaluate its predictive distributions on increasingly rotated digits. We also train models on CIFAR10 and evaluate them on data submitted to 16 different corruptions (Hendrycks and Dietterich, 2019) with 5 levels of severity each. Per severity, results are provided.

We simulate a realistic Out-of-distribution (OOD) rejection scenario (Filos et al., 2019) by jointly evaluating our models on an in-distribution and an OOD test set. We allow our methods to reject increasing proportions of the data based on predictive entropy before classifying the rest. All predictions on OOD samples are treated as incorrect. We also perform OOD detection experiments, where we evaluate methods' capacity to distinguish in-distribution and OOD points using predictive entropy.

For all datasets, we compute run times per batch of size of 256 samples on two P100 GPUs. Results are obtained as averages of 5 independent runs. Ensembles and Dropout require multiple forward passes per batch. We report the time taken for all passes to be made. For Ensembles, we also include network loading time. This is because, in most cases, keeping 5 ResNet-50's in memory is unrealistic.

A.2.4 NAS Experiments

For experiments on the spirals dataset, our input f_0 and output f_{D+1} blocks consist of linear layers. These map from input space to the selected width w and from w to the output size, respectively. Thus, selecting $d = 0 \Rightarrow b_i{=}0\,\forall i \in [1, D]$ results in a linear model. The functions applied in residual blocks, $f_i(\cdot)\,\forall i \in [1, D]$, consist of a fully connected layer followed by a ReLU activation function and Batch Normalization (Ioffe and Szegedy, 2015).

Our architecture for the image experiments uses a 5×5 convolutional layer together with a 2×2 average pooling layer as an input block f_0. No additional down-sampling layers are used. The output block, f_{D+1}, is composed of a global average pooling layer followed by a fully connected residual block, as described in the previous paragraph, and a linear layer. The function applied in the residual blocks, $f_i(\cdot)\,\forall i \in [1, D]$, matches the preactivation bottleneck residual function described by He et al. (2016b) and uses 3×3 convolutions. The outer number of channels is set to 64, and the bottleneck number is 32.

A.2.5 Active Learning Experiments

We implement batch-based active learning, with batches of 20 (for most UCI datasets) data points acquired in each query, depending on the size of the dataset. An initial training set representing a small proportion of the full data is selected uniformly at random in the first query. For all regression datasets, 80% of the data are used for training, 10% for validation and 10% for testing. The standard train-test split is used for MNIST. Details about dataset sizes, input dimensionality, and active learning specifications for each dataset are provided in Table A.6.

Table A.6: Summary of datasets and active learning specifications. 80% of the data is used for training, 10% for validation and 10% for testing.

Name	Size	Input Dim.	Init. train size	No. queries	Query size
Concrete Strength	1,030	8	50	30	20
Energy Efficiency	768	8	50	30	20
Kin8nm	8,192	8	50	30	20
Naval Propulsion	11,934	16	50	30	20
Power Plant	9,568	4	50	30	20
Protein Structure	45,730	9	50	30	20
Wine Quality Red	1,599	11	50	30	20
Yacht Hydrodynamics	308	6	20	20	10

For the regression problems, we implement a fully-connected network with residual connections, with 100 hidden nodes per layer. The networks contain 10 hidden layers for DUNs, or three hidden layers for Monte Carlo Dropout (MCDO) and MFVI. Other depths were tested for the baseline methods, with similar results to the chosen depth of three layers. We use ReLU activations, and for DUNs batch normalisation is applied after every layer (Ioffe and Szegedy, 2015). Optimisation is performed over $1,000$ iterations with full-batch gradient descent, with momentum of 0.9 and a learning rate of 10^{-4}. A weight decay value of 10^{-5} is also used. We do not implement early stopping, but the best model based on evaluation of the evidence lower bound on the validation set, is used for reporting final results. MFVI models are trained using five MC samples and the local reparameterisation trick (Kingma et al., 2015), and prediction for both MCDO and MFVI is based on 10 MC samples. For MCDO models a fixed dropout probability of 0.1 is used. Unless otherwise specified, DUNs use a uniform categorical prior over depth, while MFVI networks use a $\mathcal{N}(\mathbf{0}, I)$ prior over weights. These hyperparameter settings largely follow those used for the toy regression problems.

All experiments are repeated 40 times with different weight initialisations and train-test splits (with the exception of experiments on the Protein dataset, which are repeated only 30 times due to the cost of evaluation on the larger test set). Unless otherwise specified, we report the mean and standard deviation of the relevant metric over the repeated experiment runs.

To choose the proposal distribution temperature, we ran an ablation study. Figure A.2 shows the test NLL for DUNs using different temperatures T for the proposal distribution. The magnitude of T controls how deterministic the resulting sampling is—a a larger T corresponds to more certainly selecting the point with the highest Bayesian Active Learning by Disagreement (BALD) score, while a smaller T is closer to uniform sampling. A temperature of $T = 10$ yields the best performance for most datasets.

A.2.6 Datasets

We employ the following datasets in Chapter 3.

Regression:

- UCI with standard splits (Hernández-Lobato and Adams, 2015)
- UCI with gap splits (Foong et al., 2019)
- Flights (Hensman et al., 2013)

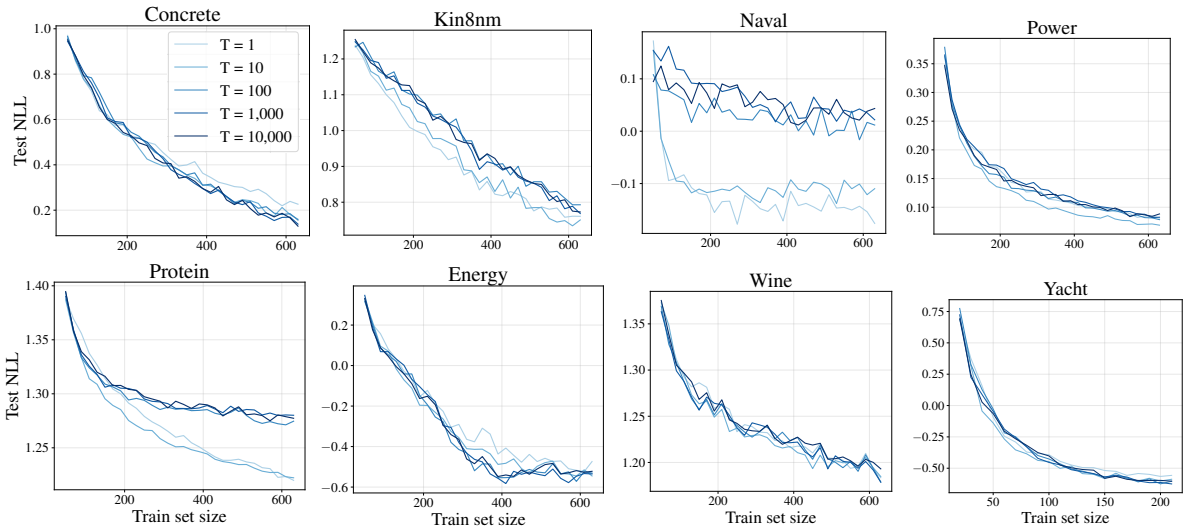


Figure A.2: Test NLLs of DUNs using a stochastic relaxation of BALD. Different temperatures of the proposal distribution are compared. Means of 40 runs of the experiments are shown; standard deviations are not plotted for clarity.

Image Classification:

- MNIST (LeCun et al., 1989)
- FashionMNIST (Xiao et al., 2017)
- KMNIST (Clanuwat et al., 2018)
- CIFAR10/100 (Krizhevsky, 2009) and Corrupted CIFAR (Hendrycks and Dietterich, 2019)
- SVHN (Netzer et al., 2011)

Appendix B

Subnetwork Inference Details

B.1 Updating the prior precision for uncertainty estimation with subnetworks

As described in Section 4.3, the linearised Laplace method can be understood as approximating our NN with a basis function linear model, where the Jacobian of the NN evaluated at $\mathbf{x}$, $\mathbf{J}(\mathbf{x}) \in \mathcal{R}^{O \times D}$ represents the feature expansion. When employing an Isotropic Gaussian prior with precision λ and for a given output dimension i, this formulation corresponds to a Gaussian Process (GP) with kernel

$$k_i(\mathbf{x}, \mathbf{x}') = \lambda^{-1} \mathbf{J}(\mathbf{x})_i \mathbf{J}(\mathbf{x}')_i^\top = \lambda^{-1} \sum_{d=1}^{D} \mathbf{J}(\mathbf{x})_{i,d} \mathbf{J}(\mathbf{x}')_{i,d}. \tag{B.1}$$

For our subnetwork model, the Jacobian feature expansion is $\mathbf{J}_S(\mathbf{x}) \in \mathbb{R}^{O \times S}$, which is a submatrix of $\mathbf{J}(\mathbf{x})$. It follows that the implied kernel will be computed in the same way as (B.1), removing $D - S$ terms from the sum. The updated prior precision $\lambda_S = \lambda \cdot S/D$ aims to maintain the magnitude of the sum, thus making the kernel corresponding to the subnetwork as similar as possible to that of the full network.

B.2 Experimental Setup

B.2.1 Toy Experiments

We train a single, 2-hidden-layer network, with 50 hidden ReLU units per layer using MAP inference until convergence. Specifically, we use Stochastic Gradient Descent (SGD) with a learning rate of 1×10^{-3}, momentum of 0.9 and weight decay of 1×10^{-4}. We use a

batch size of 512. The objective we optimise is the Gaussian log-likelihood of our data, where the mean is outputted by the network and the the variance is a hyperparameter learnt jointly with NN parameters by SGD. This variance parameter is shared among all data points. Once the network is trained, we perform post-hoc inference on it using different approaches. Since all of these involve the linearised approximation, the mean prediction is the same for all methods. Only their uncertainty estimates vary.

Note that while for this toy example, we could in principle use the full covariance matrix for the purpose of subnetwork selection, we still just use its diagonal (as described in Section 4.5) for consistency. We use Generalized Gauss-Newton (GGN) Laplace inference over network weights (not biases) in combination with the linearised predictive distribution in (4.16). Thus, all approaches considered share their predictive mean, allowing us to better compare their uncertainty estimates.

All approaches share a single prior precision of $\lambda = 3$, scaled as $\lambda_S = \lambda \cdot {}^{S}/_{D}$. We choose this prior precision such that the full covariance approach (optimistic baseline), where $\lambda_S = \lambda$, presents reasonable results. We first tried a precision of 1 and found the full covariance approach to produce excessively large error bars (covering the whole plot). A value of 3 produces more reasonable results.

Final layer inference is performed by computing the full Laplace covariance matrix and discarding all entries except those corresponding to the final layer of the NN. Results for random sub-network selection are obtained with a single sample from a scaled uniform distribution over weight choice.

B.2.2 UCI Experiments

In this experiment, our fully connected NNs have numbers of hidden layers $h_d = \{1, 2\}$ and hidden layer widths $w_d = \{50, 100\}$. For a dataset with input dimension i_d, the number of weights is given by $D = (i_d + 1)w_d + (h_d - 1)w_d^2$. Our 2-hidden-layer, 100-hidden-unit models have a weight count of the order 10^4. The non-linearity used is ReLU.

We first obtain a MAP estimate of each model's weights. Specifically, we use SGD with a learning rate of $1{\times}10^{-3}$, momentum of 0.9 and weight decay of $1{\times}10^{-4}$. We use a batch size of 512. The objective we optimise is the Gaussian log-likelihood of our data, where the mean is outputted by the network and the the variance is a hyperparameter learnt jointly with NN parameters by SGD.

For each dataset split, we set aside 15% of the train data as a validation set. We use these for early stopping training. Training runs for a maximum of 2000 epochs but early stops with a patience of 500 if validation performance does not increase. For the larger

Protein dataset, these values are 500 and 125. The weight settings which provide the best validation performance are kept.

We then perform full network GGN Laplace inference for each model. We also use our proposed Wassertein rule together with the diagonal Hessian assumption to prune every network's weight variances such that the number of variances that remain matches the size of every smaller network under consideration. The prior precision used for these steps is chosen such that the resulting predictor's log likelihood performance on the validation set is maximised. Specifically, we employ a grid search over the values: $\lambda : [0.0001, 0.001, 0.1, 0.5, 1, 2, 5, 10, 100, 1000]$. In all cases, we employ the linearised predictive in (4.16). Consequently, networks with the same number of weights make the same mean predictions. Increasing the number of weight variances considered will thus only increase predictive uncertainty.

B.2.3 Image Experiments

The results shown in Section 4.6.3 are obtained by training ResNet-18 (and ResNet-50) models using SGD with momentum. For each experiment repetition, we train 7 different models: 'MAP', 'Ours', 'Ours (Rand)', 'Stochastic Weight Averaging Gaussian (SWAG)', 'Diag-Laplace'. We train 4 additional 'Ensemble' elements, 1 network with 'Dropout', and, finally, 1 network for 'Variational Online Gauss-Newton (VOGN)'. The methods 'Ours', 'Ours (Rand)', 'SWAG', and 'Diag-Laplace' are applied post-training.

For all methods except 'VOGN' we use the following training procedure. The (initial) learning rate, momentum, and weight decay are 0.1, 0.9, and 1×10^{-4}, respectively. For 'MAP' we use 4 Nvidia P100 GPUs with a total batch size of 2048. For the calculation of the Jacobian in the subnetwork selection phase we use a single P100 GPU with a batch size of 4. For the calculation of the hessian we use a single P100 GPU with a batch size of 2. We train on 1 Nvidia P100 GPU with a batch size of 256 for all other methods. Each dataset is trained for a different number of epochs, shown in Table B.1. We decay the learning rate by a factor of 10 at scheduled epochs, also shown in Table B.1. Otherwise, all methods and datasets share hyperparameters. These hyperparameter settings are the defaults provided by `PyTorch` for training on ImageNet. We found them to perform well across the board. We report results obtained at the final training epoch. We do not use a separate validation set to determine the best epoch as we found ResNet-18 and ResNet-50 to not overfit with the chosen schedules.

For 'Dropout', we add dropout to the standard ResNet-50 model (He et al., 2016a) in between the 2^{nd} and 3^{rd} convolutions in the bottleneck blocks. This approach follows Zagoruyko and Komodakis (2016) and Ashukha et al. (2020), who add dropout

Table B.1: Per-dataset training configuration for image experiments.

Dataset	No. Epochs	LR Schedule
MNIST	90	40, 70
CIFAR10	300	150, 225

in between the two convolutions of a WideResNet-50's basic block. Following Antorán et al. (2020), we choose a dropout probability of 0.1, as they found it to perform better than the value of 0.3 suggested by Ashukha et al. (2020). We use 16 Monte Carlo (MC) samples for predictions. 'Ensemble' uses 5 elements for prediction. Ensemble elements differ from each other in their initialisation, which is sampled from the He initialisation distribution (He et al., 2015). We do not use adversarial training as, in line with Ashukha et al. (2020), we do not find it to improve results. For 'VOGN' we use the same procedure and hyper-parameters as used by Osawa et al. (2019) in their CIFAR10 experiments, with the exception that we use a learning rate of 1×10^{-3} as we we found a value of 1×10^{-4} not to result in convergence. We train on a single Nvidia P100 GPU with a batch size of 256. See the authors' GitHub for more details: `github.com/team-approx-bayes/dl-with-bayes/blob/master/distributed/classification/configs/cifar10/resnet18_vogn_bs256_8gpu.json`.

We modify the standard ResNet-50 and ResNet-18 architectures such that the first 7×7 convolution is replaced with a 3×3 convolution. Additionally, we remove the first max-pooling layer. Following Goyal et al. (2017), we zero-initialise the last batch normalisation layer in residual blocks so that they act as identity functions at the start of training.

At test time, we tune the prior precision used for 'Ours', 'Diag-Laplace' and 'SWAG' approximation on a validation set for each approach individually, as in Ritter et al. (2018); Kristiadi et al. (2020). We use a grid search from 1×10^{-4} to 1×10^{4} in logarithmic steps, and then a second, finer-grained grid search between the two best-performing values (again with logarithmic steps).

B.2.4 Datasets

The 1d toy dataset used in Section 4.6.1 was taken from Antorán et al. (2020). We obtained it from the authors' GitHub repo: `https://github.com/cambridge-mlg/DUN`. Table B.2 summarises the datasets used in Section 4.6.2.

We employ the Wine, Kin8nm and Protein datasets, together with their gap variants, because we find our models' performance to be most dependent on the quality of the

estimated uncertainty here. On most other commonly used UCI regression datasets (Hernández-Lobato and Adams, 2015) we find increased uncertainty to hurt LL performance. In other words, the predictions made when using the MAP setting of the weights are better than those from any Bayesian ensemble. For the standard splits (Hernández-Lobato and Adams, 2015) 90% of the data is used for training and 10% for validation. For the gap splits (Foong et al., 2019) a split is obtained per input dimension by ordering points by their values across that dimension and removing the middle 33% of the points. These are used for validation.

The datasets used for our image experiments are outlined in Table B.3.

Table B.2: Datasets from tabular regression used in Section 4.6.2

Dataset	N Train	N Val (15% train)	N Test	Splits	Output Dim	Output Type	Input Dim	Input Type
Wine	1223	216	160	20	1	Continous	11	Continous
Wine Gap	906	161	532	11	1	Continous	11	Continous
Kin8nm	6267	1106	819	20	1	Continous	8	Continous
Kin8nm Gap	4642	820	2730	8	1	Continous	8	Continous
Protein	34983	6174	4573	5	1	Continous	9	Continous
Protein Gap	25913	4573	15244	9	1	Continous	9	Continous

Table B.3: Summary of image datasets. The test and train set sizes are shown in brackets, e.g., (test & train).

Name	Size	Input Dim.	No. Classes	No. Splits
MNIST (LeCun et al., 1989)	70,000 (60,000 & 10,000)	784 (28 × 28)	10	2
Fashion-MNIST (Xiao et al., 2017)	70,000 (60,000 & 10,000)	784 (28 × 28)	10	2
CIFAR10 (Krizhevsky, 2009)	60,000 (50,000 & 10,000)	3072 (32 × 32 × 3)	10	2
SVHN (Netzer et al., 2011)	99,289 (73,257 & 26,032)	3072 (32 × 32 × 3)	10	2

Appendix C

Sparse MoE Details

C.1 Experiment Settings

C.1.1 ViT Model Specifications

Following Dosovitskiy et al. (2021), we recall the specifications of the Vision Transformer (ViT) models of different scales in Table C.1.

Table C.1: Specifications of ViT-S, ViT-B, ViT-L and ViT-H.

	Hidden dimension	MLP dimension	# layers
Small	512	2048	8
Base	768	3072	12
Large	1024	4096	24
Huge	1280	5144	32

C.1.2 Upstream Setting

For all our upstream experiments, we scrupulously follow the setting described in Riquelme et al. (2021), see their Section B.2 in their appendix. For completeness, we just recall that S/32 models are trained for 5 epochs while B/{16, 32} and L/32 models are trained for 7 epochs. For L/16 models, both 7 and 14 epochs can be considered (Dosovitskiy et al., 2021; Riquelme et al., 2021); we opted for 7 epochs given the breadth of our experiments. Finally, the H/14 model is trained for 14 epochs.

In particular, the models are all trained on JFT-300M (Sun et al., 2017). This dataset contains about 305M training and 50 000 validation images. The labels have a hierarchical structure, with a total of 18 291 classes, leading to 1.89 labels per image on average.

C.1.3 Downstream Setting

During fine-tuning, there are a number of *common* design choices we apply. In particular:

- Image resolution: 384.
- Clipping gradient norm at: 10.0.
- Optimizer: Stochastic Gradient Descent (SGD) with momentum (using half-precision, $\beta = 0.9$).
- Batch size: 512.
- For Vision-MoE (V-MoE) models, we fine-tune with capacity ratio $C = 1.5$ and evaluate with $C = 8$.

We use the following train/validation splits depending on the dataset:

Dataset	Train Dataset Fraction	Validation Dataset Fraction
ImageNet	99%	1%
CIFAR10	98%	2%
CIFAR100	98%	2%
Oxford-IIIT Pets	90%	10%
Oxford Flowers-102	90%	10%

All of the above design choices follow from Riquelme et al. (2021) and Dosovitskiy et al. (2021).

C.1.4 Hyperparameter Sweep for Fine-tuning

In all our fine-tuning experiments, we use the sweep of hyperparameters described in Table C.2. We use the recommendations from Dosovitskiy et al. (2021) and Riquelme et al. (2021), further considering several factors {0.5, 1.0, 1.5, 2.0} to sweep over different numbers of steps. Riquelme et al. (2021) use a half schedule (with the factor 0.5) while Dosovitskiy et al. (2021) take the factor 1.0.

C.1.5 Details about the (Linear) Few-shot Evaluation

We follow the evaluation methodology proposed by Dosovitskiy et al. (2021); Riquelme et al. (2021), which we recall for completeness. Let us rewrite our model f with parameters

Table C.2: Hyperparameter values for fine-tuning on different datasets. Compared with Dosovitskiy et al. (2021) and Riquelme et al. (2021), we further consider several factors {0.5, 1.0, 1.5, 2.0} to sweep over different numbers of steps. We also apply a dropout rate of 0.1 to the expert Multi-Layer Perceptrons (MLPs).

Dataset	Steps	Base LR
ImageNet	20 000 × {0.5, 1.0, 1.5, 2.0}	{0.0024, 0.003, 0.01, 0.03}
CIFAR10	5 000 × {0.5, 1.0, 1.5, 2.0}	{0.001, 0.003, 0.01, 0.03}
CIFAR100	5 000 × {0.5, 1.0, 1.5, 2.0}	{0.001, 0.003, 0.01, 0.03}
Oxford-IIIT Pets	500 × {0.5, 1.0, 1.5, 2.0}	{0.001, 0.003, 0.01, 0.03}
Oxford Flowers-102	500 × {0.5, 1.0, 1.5, 2.0}	{0.001, 0.003, 0.01, 0.03}

$\boldsymbol{\theta} = \{\boldsymbol{Q}, \boldsymbol{\theta}'\}$ as

$$f(\boldsymbol{x}; \boldsymbol{\theta}) = \texttt{softmax}(\boldsymbol{Q}\phi(\boldsymbol{x}; \boldsymbol{\theta}'))$$

where $\boldsymbol{Q} \in \mathbb{R}^{C \times S}$ corresponds to the parameters of the last layer of f with the S-dimensional representation $\phi(\boldsymbol{x}; \boldsymbol{\theta}') \in \mathbb{R}^S$.

In linear few-shot evaluation, we construct a linear classifier to predict the target labels (encoded as one-hot vectors) from the S-dimensional feature vectors induced by $\phi(\cdot; \boldsymbol{\theta}')$; see Chapter 5 in Hastie et al. (2017) for more background about this type of linear classifier. This evaluation protocol makes it possible to evaluate the quality of the representations ϕ learned by f.

While Dosovitskiy et al. (2021); Riquelme et al. (2021) essentially focus on the quality of the representations learned upstream on JFT by computing the (linear) few-shot accuracy on ImageNet, we are interested in the representations after fine-tuning on ImageNet. As a result, we consider a collection of 8 few-shot datasets (that do not contain ImageNet):

- Caltech-UCSD Birds 200 (Wah et al., 2011) with 200 classes,
- Caltech 101 (Bansal et al., 2021) with 101 classes,
- Cars196 (Krause et al., 2013) with 196 classes,
- CIFAR100 (Krizhevsky, 2009) with 100 classes,
- Colorectal histology (Kather et al., 2016) with 8 classes,
- Describable Textures Dataset (Cimpoi et al., 2014) with 47 classes,
- Oxford-IIIT pet (Parkhi et al., 2012) with 37 classes and

- UC Merced (Yang and Newsam, 2010) with 21 classes.

In the experiments, we compute the few-shot accuracy for each of the above datasets and we report the averaged accuracy over the datasets, for various number of shots in $\{1, 5, 10, 25\}$. As commonly defined in few-shot learning, we understand by s shots a setting wherein we have access to s training images per class label in each of the datasets.

To account for the different scales of accuracy across the 8 datasets, we also tested to compute a weighted average, normalising by the accuracy of a reference model (ViT-B/32). This is reminiscent of the normalisation carried out in Hendrycks and Dietterich (2019) according to the score of AlexNet. We found the conclusions with the standard average and weighted average to be similar.

For an ensemble with M members, we have access to M representations $\{\phi(\boldsymbol{x}; \boldsymbol{\theta}'_m)\}_{m=1}^M$ for a given input $\boldsymbol{x}$. We concatenate the M representations $\{\phi(\boldsymbol{x}; \boldsymbol{\theta}'_m)\}_{m=1}^M$ into a single "joint" feature vector in $\mathbb{R}^{M \cdot S}$, remembering that each $\phi(\boldsymbol{x}; \boldsymbol{\theta}'_m) \in \mathbb{R}^S$. We then train a *single* linear classifier to predict the target labels from the "joint" feature vectors.

C.1.6 List of Datasets

For completeness, in addition to the few-shot datasets listed in Section C.1.5, we list the datasets used for downstream training and evaluation in this work.

- ImageNet (ILSVRC2012) (Deng et al., 2009) with 1000 classes and 1281167 training examples.
- ImageNet-C (Hendrycks and Dietterich, 2019), an ImageNet test set constructed by applying 15 different corruptions at 5 levels of intensity to the original ImageNet test set. (We report the mean performance over the different corruptions and intensities.)
- ImageNet-A (Hendrycks et al., 2021), an ImageNet test set constructed by collecting new data and keeping only those images which a ResNet-50 classified incorrectly.
- ImageNet-V2 (Recht et al., 2019), an ImageNet test set independently collected using the same methodology as the original ImageNet dataset.
- CIFAR10 (Krizhevsky, 2009) with 10 classes and 50000 training examples.
- CIFAR10-C (Hendrycks and Dietterich, 2019), a CIFAR10 test set constructed by applying 15 different corruptions at 5 levels of intensity to the original CIFAR10 test set. (We report the mean performance over the different corruptions and intensities.)

- CIFAR100 (Krizhevsky, 2009) with 100 classes and training 50000 examples.
- Oxford Flowers 102 (Nilsback and Zisserman, 2008) with 102 classes and 1020 training examples.
- Oxford-IIIT pet (Parkhi et al., 2012) with 37 classes and 3680 training examples.
- SVHN (Netzer et al., 2011) with 10 classes.
- Places365 (Zhou et al., 2017) with 365 classes.
- Describable Textures Dataset (DTD) (Cimpoi et al., 2014) with 47 classes.

C.1.7 Sparse MoEs meet Ensembles Experimental Details

The setup for the experiments in Figures C.2 and 5.2 differs slightly from the other experiments in this work. Specifically, while for all other experiments we used upstream V-MoE checkpoints with $(K, E) = (2, 32)$, for these experiments we matched the upstream and downstream checkpoints. We did this to avoid a checkpoint mismatch as a potential confounder in our results.

C.1.8 Multiple Predictions without Tiling or Partitioning Details

The naive multi-pred method presented in Section 5.4.2 was trained in almost the same manner as the vanilla V-MoE, the only difference being the handling of multiple predictions. This was accomplished by using the average ensemble member cross entropy as described for Efficient Ensemble of Experts (E^3) in Section C.3. In contrast, in order to compute the evaluation metrics presented in Table 5.4, we first averaged predictions of the model and then used the average prediction when calculating each metric.

C.2 Compatibility and Adaptation of the Upstream Checkpoints

Throughout the Chapter 5, we make the assumption that we can start from existing checkpoints of ViT and V-MoE models (trained on JFT-300M; see Section C.1.2). We next describe how we can use those checkpoints for the fine-tuning of the extensions of ViT and V-MoE that we consider in this chapter.

In all our experiments that involve V-Mixtures of Experts (MoEs), we consider checkpoints with $K = 2$ and $E = 32$, which is the canonical setting advocated by Riquelme et al. (2021).

C.2.1 Efficient Ensemble of Experts

In the case of E^3, the set of parameters is identical to that of a V-MoE model. In particular, neither the tiled representation nor the partitioning of the experts transforms the set of parameters.

To deal with the fact that the single routing function $\texttt{gate}_K(\boldsymbol{W}\cdot)$ of a V-MoE becomes separate routing functions $\{\texttt{gate}_K(\boldsymbol{W}_m\cdot)\}_{m=1}^M$, one for expert subset $\mathcal{E}_m$, we simply slice row-wise $\boldsymbol{W} \in \mathbb{R}^{E\times D}$ into the M matrices $\boldsymbol{W}_m \in \mathbb{R}^{(E/M)\times D}$.

C.2.2 Batch Ensembles (BE)

We train Batch Ensemble (BE) starting from ViT checkpoints, which requires to introduce downstream-specific parameters. Following the design of V-MoEs, we place the batch-ensemble layers in the MLP layers of the Transformer.

Let us consider a dense layer in one of those MLPs, with parameters $\boldsymbol{U} \in \mathbb{R}^{D\times L}$, in absence of a bias term. In BE, the parametrization of each ensemble member has the following structure $\boldsymbol{U}_m = \boldsymbol{U} \circ (\boldsymbol{r}_m \boldsymbol{s}_m^\top)$ where $\{\boldsymbol{r}_m\}_{m=1}^M$ and $\{\boldsymbol{s}_m\}_{m=1}^M$ are respectively D- and L-dimensional vectors.

A standard ViT checkpoint provides pre-trained parameters for $\boldsymbol{U}$. We then introduce $\{\boldsymbol{r}_m\}_{m=1}^M$ and $\{\boldsymbol{s}_m\}_{m=1}^M$ at fine-tuning time, following the random initialization schemes proposed in Wen et al. (2020); see details in the hyperparameter sweep for BE in Section C.7.1.

C.2.3 MIMO

We train Multi-input Multi-output (MIMO) models from V-MoE checkpoints. The only required modifications are to the input and output parameters of the checkpoints. The linear input embedding must be modified to be compatible with input images containing M times as many channels, as required by the multiple-input structure of MIMO. Similarly, the final dense layer in the classification head must be modified to have M times more output units, following the multiple-output structure in MIMO.

Concretely, the embedding weight $\boldsymbol{W}_{\text{in}} \in \mathbb{R}^{H\times W\times 3\times D}$ is replicated in the third (channel) dimension, resulting in $\boldsymbol{W}_{\text{MIMO,in}} \in \mathbb{R}^{H\times W\times 3\cdot M\times D}$, where H and W are the height

and width of the convolution and D is the hidden dimension of the ViT family (specified in Table C.1). The output layer weight $\boldsymbol{W}_{\text{out}} \in \mathbb{R}^{D \times C}$ is replicated in the second (output) dimension, resulting in $\boldsymbol{W}_{\text{MIMO,out}} \in \mathbb{R}^{H \times C \cdot M}$, where C is the number of classes. The output layer bias $\boldsymbol{b}_{\text{out}} \in \mathbb{R}^{C}$ is replicated resulting in $\boldsymbol{b}_{\text{MIMO,out}} \in \mathbb{R}^{C \times M}$. Finally, in order to preserve the magnitude of the activation for these layers, $\boldsymbol{W}_{\text{MIMO,in}}$ and $\boldsymbol{W}_{\text{MIMO,out}}$ are scaled by $1/M$.

C.3 Implementation Details of Efficient Ensemble of Experts

We provide details about the training loss and the regulariser used by E^3. We also discuss the memory requirements compared to V-MoE.

C.3.1 Training Loss

Since E^3 outputs M predictions $\{f(\boldsymbol{x}; \boldsymbol{\theta}_m)\}_{m=1}^{M}$ for a given input $\boldsymbol{x}$, we need to adapt the choice of the training loss $\mathcal{L}$ accordingly. Following the literature on efficient ensembles (Wen et al., 2020; Dusenberry et al., 2020a; Wenzel et al., 2020b), we choose the average ensemble-member cross entropy

$$\mathcal{L}(y, \boldsymbol{x}; \boldsymbol{\theta}) = \frac{1}{M} \sum_{m=1}^{M} \text{cross-entropy}(y, f(\boldsymbol{x}; \boldsymbol{\theta}_m))$$

instead of other alternatives, such as the ensemble cross-entropy

$$\text{cross-entropy}\Big(y, \frac{1}{M} \sum_{m=1}^{M} f(\boldsymbol{x}; \boldsymbol{\theta}_m)\Big)$$

that was observed to generalise worse (Dusenberry et al., 2020a).

C.3.2 Auxiliary Losses

Inspired by previous applications of sparse MoEs in NLP (Shazeer et al., 2017), Riquelme et al. (2021) employ regularisers, also referred to as *auxiliary losses*, to guarantee a balanced usage of the E experts. Two auxiliary losses—the importance and load losses, see Appendix A in Riquelme et al. (2021) for their formal definitions—are averaged together to form the final regularisation term that we denote by Ω.

As a reminder, let us recall the notation of the routing function

$$\boldsymbol{h} \in \mathbb{R}^D \mapsto \texttt{gate}_K(\boldsymbol{W}\boldsymbol{h}) = \texttt{top}_K(\texttt{softmax}(\boldsymbol{W}\boldsymbol{h} + \sigma\boldsymbol{\varepsilon})) \in \mathbb{R}^E,$$

with $\boldsymbol{W} \in \mathbb{R}^{E\times D}$ and $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. Consider a batch of B inputs $\{\boldsymbol{h}_i\}_{i=1}^B$ that we represent by $\boldsymbol{H} \in \mathbb{R}^{B\times D}$. Finally, let us define

$$\boldsymbol{A} = \boldsymbol{H}\boldsymbol{W}^\top + \sigma\boldsymbol{\varepsilon}_{B\times E} \in \mathbb{R}^{B\times E},$$

where we emphasise that $\boldsymbol{\varepsilon}_{B\times E}$ is a matrix of Gaussian noise entries in $\mathbb{R}^{B\times E}$. The regularisation term Ω used by Riquelme et al. (2021) can be seen as a function that depends on $\boldsymbol{A}$ and $\boldsymbol{H}\boldsymbol{W}^\top$.

In the context of efficient ensemble of experts, the set of E experts is partitioned into M subsets of E/M experts, denoted by $\cup_{m=1}^M \mathcal{E}_m$; see Section 5.4.1. With the introduction of the M routing functions $\{\texttt{gate}_K(\boldsymbol{W}_m\cdot)\}_{m=1}^M$ with each $\boldsymbol{W}_m \in \mathbb{R}^{(E/M)\times D}$, the matrix $\boldsymbol{A}$ becomes accordingly partitioned into $\{\boldsymbol{A}_m\}_{m=1}^M$ where each $\boldsymbol{A}_m \in \mathbb{R}^{B\times(E/M)}$.

Since we want to enforce a balanced usage of the E/M experts in each subset $\mathcal{E}_m$ of the experts, we thus redefine the regularisation as the average regularisation separately applied to each part of the partition

$$\Omega_{\text{partition}}(\boldsymbol{A}, \boldsymbol{H}\boldsymbol{W}^\top) = \frac{1}{M}\sum_{m=1}^{M} \Omega(\boldsymbol{A}_m, \boldsymbol{H}\boldsymbol{W}_m^\top).$$

We found this option to work better in practice. To guarantee a fair comparison, we also applied $\Omega_{\text{partition}}$ to the "Only partitioning" model in the ablation study of Section 5.4.2.

Following Riquelme et al. (2021), the regularisation parameter controlling the strength of $\Omega_{\text{partition}}$ was set to 0.01 throughout the experiments.

C.3.3 Memory Requirements versus V-MoE

Due to tiling, E^3 requires more memory than V-MoE. To be concrete, the memory complexity of V-MoE can be decomposed into two terms

$$\mathcal{O}\left(\texttt{memory}_{\text{params}} + \texttt{memory}_{\text{activations}}\right),$$

for the forward and backward passes, respectively. For E^3, the complexity becomes

$$\mathcal{O}\left(\texttt{memory}_{\text{params}} + \texttt{memory}_{\text{activations}} \times \frac{L_{\text{before}} + L_{\text{after}} \times M}{L_{\text{total}}}\right),$$

where M is the ensemble size, and L_{before}, L_{after}, and L_{total} are the number of layers before tiling, after tiling, and in total, respectively. Importantly, neither $\texttt{memory}_{\text{params}}$ nor $\texttt{memory}_{\text{activations}}$ depend on M. Thanks to the "last-n" setting employed in the paper, we have $L_{\text{after}} \ll L_{\text{before}}$, and thus the increase in memory due to tiling remains mild. More concretely, for ViT-L, we have $L_{\text{total}} = 24$, $L_{\text{before}} = 21$, and $L_{\text{after}} = 3$ (with MoE layers placed at layers 22 and 24). Thus, for an ensemble of size $M = 2$, E^3 would only increase $\texttt{memory}_{\text{activations}}$ by 12.5%, while leaving $\texttt{memory}_{\text{params}}$ unchanged.

C.4 Efficient Ensemble of Experts and V-MoE Relative Improvements per ViT Family

In Section 5.5 we claim that E^3 performs best at the largest scale. In this section, we motivate that claim in more detail. Specifically, we consider two metrics of improvement in performance. Firstly, we consider the percentage improvement in Negative Log Likelihood (NLL) for both E^3 and V-MoE versus vanilla ViT. Secondly, we consider a normalised version of this improvement. We consider this second metric to take into account the "difficulty" in further improving the NLL of larger ViT family models. Intuitively, the larger the ViT family, the better the corresponding NLL will be, and the more difficult it will be to improve on that NLL.

The normalisation we apply is based on the gradient of the NLL with respect to Floating Point Operations Per Second (FLOPs). Indeed, this gradient captures the typical variation of NLL at a particular amount of FLOPs. The ratio of this gradient at the FLOPs values (i.e., the instantaneous change in NLL at those FLOPs values) for two ViT families is a measure of the relative difficulty in increasing the NLL. Thus, we can use this ratio to normalise our results. To be more concrete, let us define the mapping

$$\text{NLL} = \varphi(\text{FLOPs}) \quad \text{and its derivative} \quad \varphi'(\text{FLOPs}) = \frac{d\varphi(\text{FLOPs})}{d\text{FLOPs}}.$$

We estimate φ and its gradient by fitting a linear model to the $(\text{NLL}, \text{FLOPs})$ pairs for each ViT family, using the data of the standard ViT models we trained. We use feature expansion $[1.0, \log(\text{FLOPs}), \log(\text{FLOPs})^2, \log(\text{FLOPs})^3]$ and solve for the parameters of the linear model via ordinary least squares. We determine the gradient of this function at each FLOPs value using automatic differentiation in `JAX`. See Figure C.1 for the resulting fit and an indication of the gradients.

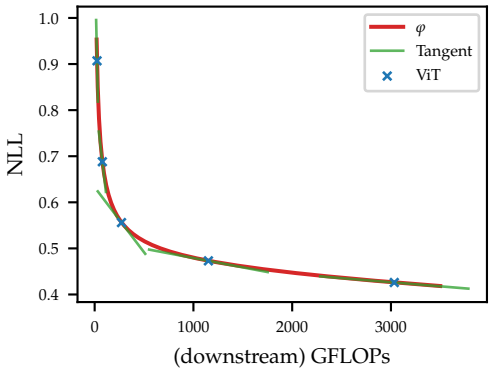


Figure C.1: Estimated φ compared to the ImageNet NLL values for our ViT models. We also include the tangent at the points corresponding to each ViT model to indicate the gradients at those points.

The normalised values are calculated as:

$$\text{Normalised improvement}(v) = \text{improvement}(v) \times \frac{\varphi'(\text{FLOPs}_{\text{H/14}})}{\varphi'(\text{FLOPs}_v)}, \tag{C.1}$$

where v is one of the ViT families, i.e., S/32, B/32, L/32, L/16, or H/14. Note that this normalisation leaves the improvement for H/14 the same. We tried to normalise with respect to other choices of ViT family, different from H/14. Our conclusions are robust in the sense that both the ordering and the monotonic behaviour with respect to scale are preserved. Using the ratio for normalisation also has the advantage that the normalisation is less sensitive to the particular parameterisation of φ.

Table C.3: Percentage improvements in NLL for E^3 with $(K, M) = (1, 2)$ and V-MoE with $K = 1$ vs. ViT for families of increasing size. The top two rows show normalised improvements, see (C.1), which take into consideration the increased difficulty of improving NLL for larger ViT families whose performance is beginning to saturate. The bottom two rows are the original percentage improvements without normalisation.

		S/32	B/32	L/32	L/16	H/14
Normalised	E^3 vs. ViT	0.02%	0.09%	0.24%	2.35%	4.27%
	V-MoE vs. ViT	0.01%	0.06%	-0.04%	0.89%	0.02%
Not normalised	E^3 vs. ViT	9.82%	9.53%	3.76%	5.38%	4.27%
	V-MoE vs. ViT	7.98%	6.62%	-0.60%	2.05%	0.02%

Table C.3 shows both the difficulty-normalised and original improvements (without normalisation). Looking first at the original improvements, we can see that while both E^3 and V-MoE have smaller improvements over ViT for larger families, E^3's improvements

decrease more slowly. Furthermore, by comparing the normalised improvements, we see that E^3's improvements actually *grow* monotonically when taking difficulty into account. This is not the case for V-MoE.

C.5 From Batch Ensembles to Sparse MoEs

Wen et al. (2020) have shown that, given a batch of B inputs $\boldsymbol{X} \in \mathbb{R}^{B \times P}$, a single forward pass can efficiently compute the predictions of all the ensemble members $\{f(\boldsymbol{X}; \boldsymbol{\theta}_m)\}_{m=1}^M$. By appropriately tiling the inputs of the network $\boldsymbol{X}_{\text{tiled}} \in \mathbb{R}^{(M \cdot B) \times P}$ by a factor M, each internal operation per ensemble member can then be vectorised.

We take the previous example of a dense layer with parameters $\boldsymbol{U} \in \mathbb{R}^{D \times L}$ and we assume the layer receives the tiled inputs $\{\boldsymbol{H}_m\}_{m=1}^M$ where $\boldsymbol{H}_m \in \mathbb{R}^{B \times D}$. We need to compute for each ensemble member $\boldsymbol{H}_m \boldsymbol{U}_m = \boldsymbol{H}_m[\boldsymbol{U} \circ (\boldsymbol{r}_m \boldsymbol{s}_m^\top)]$. Denoting by $\boldsymbol{h}_{i,m} \in \mathbb{R}^D$ the i-th input in $\boldsymbol{H}_m$, we have

$$\boldsymbol{h}_{i,m}^\top \boldsymbol{U}_m = \sum_{e=1}^{E} g_e(\boldsymbol{h}_{i,m}) \cdot \texttt{expert}_e(\boldsymbol{h}_{i,m}) \quad \text{with } M = E, \quad \begin{cases} g_e(\boldsymbol{h}_{i,m}) = 1 \text{ if } e = m, \\ g_e(\boldsymbol{h}_{i,m}) = 0 \text{ otherwise} \end{cases} \tag{C.2}$$

and $\texttt{expert}_e(\boldsymbol{z}) = \boldsymbol{z}^\top[\boldsymbol{U} \circ (\boldsymbol{r}_e \boldsymbol{s}_e^\top)]$. Although (C.2) may appear as a convoluted way of writing the operations in batch ensembles, it unveils a connection with (5.1). Indeed, operations in batch ensembles can be seen as a specific sparse MoE, e.g., with binary routing weights depending only on the position in the tiled inputs. While Wen et al. (2020) primarily tiled the inputs for the sake of efficiency, it also induces some form of conditional computation, an insight that we exploit in E^3.

C.6 Batch Ensembles versus Efficient Ensemble of Experts

Table C.4: Summary of the differences between BE and E^3.

	Shared Parameters	Unique Parameters	Tiling	Training Epochs
BE	Kernels in each linear layer	"Fast-weights" and biases	Start of the model	Typically ≈50% more
E^3	All parameters outside MoE layer	Experts in subset $\mathcal{E}_m$	Before 1$^{\text{st}}$ MoE layer	Unchanged

While E^3 is inspired in several ways by BE, it has also been specialised to sparse MoEs. This leads to a number of significant differences between the two algorithms. These differences are summarised in Table C.4, and we elaborate upon them here:

- **Shared parameters:** In BE, the shared parameters are the kernels in each linear (e.g., dense or convolution) layer. This makes up the majority of the parameters in BE. In E^3, the shared parameters are all of those not found in the expert layers.
- **Ensemble member specific parameters:** BE uses pairs of "fast-weights" $\{\boldsymbol{r}_m, \boldsymbol{s}_m\}$ and bias terms for each linear layer. E^3 uses the parameters of all the E/M experts in the m-th subset $\mathcal{E}_m$ and the router for those expert, in each MoE layer. This results in a much larger set of ensemble member-specific parameters, and thus a higher level of predictive diversity (see Section 5.4.2).
- **Tiling:** In BE, tiling is performed at the very beginning of the model, regardless of the model's internal structure. Tiling in E^3 occurs before the first MoE layer, thus saving redundant computation for "Last-n" V-MoE.
- **Number of training epochs:** BE usually requires 50% more training epochs than a vanilla model. E^3 requires the same number of training epochs as V-MoE.

C.7 Efficient Ensemble Comparisons

In this section, we compare efficient ensemble of experts (E^3) to several popular efficient ensemble approaches, namely MIMO (Havasi et al., 2020), BE (Wen et al., 2020), and Monte Carlo (MC) Dropout (Gal and Ghahramani, 2016).

Table C.5 reports the ImageNet performance of those different techniques, when all models are based on a ViT-B/32 architecture. We start by highlighting the most salient conclusions of the experiment and defer to the next subsections the descriptions of the different competing techniques.

We make the following observations:

- BE built upon ViT improves the performance of ViT in terms of NLL, classification error and Expected Calibration Error (ECE). However, the resulting increase in FLOPs makes BE a less viable option compared to E^3.
- MC Dropout V-MoE is on par with standard V-MoE in terms of NLL and classification error, while it improves the ECE. For all values of K, we observe that the performance tends to improve as the number of samples, i.e., M, increases.

Table C.5: ImageNet performance of different efficient ensemble approaches. The table reports the means ± standard errors over 8 replications. All models have a ViT-B/32 architecture. K stands for the sparsity in V-MoEs, M denotes the ensemble size while "BR" corresponds to the batch repetition in MIMO (Havasi et al., 2020).

	K	M	NLL ↓	Error ↓	ECE ↓	KL ↑	GFLOPs ↓
ViT	–	–	0.688 ± 0.003	18.65 ± 0.08	0.022 ± 0.000	–	78.0
BE ViT	–	2	0.682 ± 0.003	18.47 ± 0.05	0.021 ± 0.000	0.040 ± 0.001	97.1
	–	4	0.675 ± 0.003	18.40 ± 0.09	0.017 ± 0.000	0.035 ± 0.001	135.4
V-MoE	1	–	0.642 ± 0.002	16.90 ± 0.05	0.029 ± 0.001	–	82.4
	2	–	0.638 ± 0.001	16.76 ± 0.05	0.033 ± 0.001	–	94.9
	4	–	0.636 ± 0.001	16.70 ± 0.04	0.034 ± 0.001	–	120.1
	8	–	0.635 ± 0.002	16.72 ± 0.06	0.028 ± 0.001	–	170.4
MC Dropout V-MoE	1	2	0.648 ± 0.002	17.10 ± 0.05	0.019 ± 0.001	0.046 ± 0.000	97.2
	1	4	0.641 ± 0.002	16.96 ± 0.05	0.017 ± 0.001	0.046 ± 0.001	135.6
	2	2	0.642 ± 0.002	16.94 ± 0.04	0.021 ± 0.001	0.046 ± 0.001	113.7
	2	4	0.634 ± 0.001	16.80 ± 0.03	0.020 ± 0.000	0.046 ± 0.001	168.6
	4	2	0.639 ± 0.002	16.91 ± 0.06	0.022 ± 0.001	0.045 ± 0.001	146.7
MIMO V-MoE (BR=1)	2	2	0.636 ± 0.002	16.97 ± 0.04	0.028 ± 0.001	0.000 ± 0.000	96.3
	2	4	0.672 ± 0.001	17.72 ± 0.04	0.037 ± 0.000	0.001 ± 0.000	99.0
MIMO V-MoE (BR=2)	2	2	0.638 ± 0.001	17.14 ± 0.03	0.031 ± 0.000	0.001 ± 0.000	192.6
	2	4	0.665 ± 0.002	17.38 ± 0.04	0.038 ± 0.000	0.000 ± 0.000	198.1
E^3	1	2	0.622 ± 0.001	16.70 ± 0.03	0.018 ± 0.000	0.217 ± 0.003	105.9
	1	4	0.624 ± 0.001	16.99 ± 0.03	0.013 ± 0.000	0.164 ± 0.001	153.0
	2	2	0.612 ± 0.001	16.49 ± 0.02	0.013 ± 0.000	0.198 ± 0.003	131.1
	2	4	0.620 ± 0.001	16.86 ± 0.02	0.015 ± 0.000	0.170 ± 0.001	203.3
	4	2	0.611 ± 0.001	16.45 ± 0.03	0.014 ± 0.000	0.193 ± 0.003	181.4

However, already for M in $\{2, 4\}$, the resulting increase in FLOPs makes MC Dropout V-MoE a less favourable option compared to E^3.

- Perhaps surprisingly (see detailed investigations in Section C.7.3), MIMO V-MoE does not lead to improvements compared with V-MoE. In fact, for higher ensemble sizes, MIMO V-MoE results in worse performance than standard V-MoE. Moreover, increasing the batch repetition parameter of MIMO ("BR" in Table C.5) further worsens the results. Interestingly, we can see that MIMO does not manage to produce diverse predictions, as illustrated by the small values of KL.

- E^3 offers the best performance vs. FLOPs trade-offs, e.g., when looking at $(K, M) = (1, 2)$ and $(K, M) = (2, 2)$. We notably observe that the diversity of the predictions in E^3 is orders of magnitude larger than that of the other ensemble approaches.

We briefly recall the optimisation explained in Section 5.4.1 to save redundant computations: In the "last-n" setting of Riquelme et al. (2021), it is sufficient to tile the

representations only when entering the first MoE layer/dropout layer/batch-ensemble layer for respectively E^3/MC Dropout V-MoE/BE. We apply this optimisation to all the efficient ensemble methods.

C.7.1 Batch Ensembles

Following the design of V-MoEs, we place the batch-ensemble layers in the MLP layers of the Transformer, following the "last-n" setting of Riquelme et al. (2021); see Section 5.2.1.

The vectors of the rank-1 parametrisation introduced at fine-tuning time (see Section C.2) need to be initialised and optimised. Following the recommendation from Wen et al. (2020), we consider the following hyperparameters in addition to the common sweep described in Table C.2:

- **Initialization:** Either a random sign vector with entries in $\{-1, 1\}$ independently drawn with probability $\frac{1}{2}$ or a random Gaussian vector with entries independently drawn from $\mathcal{N}(1, 0.5)$.
- **Learning-rate scale factor:** The vectors of the rank-1 parametrisation are updated with a learning rate scaled by a factor in $\{0.5, 1, 2\}$.

C.7.2 MC Dropout V-MoEs

For MC Dropout V-MoE, we take the available fine-tuned V-MoEs and enable dropout at prediction time. Indeed, as described in Table C.2, all V-MoE models already have a 0.1 dropout rate in the experts.

C.7.3 MIMO V-MoEs

Following Havasi et al. (2020) we consider two MIMO-specific hyperparameters, in addition to the hyperparameters listed in Table C.2:

- **Input replication probability:** $\{0.5, 0.625, 0.75\}$
- **Batch repetitions:** $\{1, 2\}$

Our preliminary investigations also considered lower input repetition probabilities and higher batch repetitions. However, lower input repetition probabilities tended to result in poorer performance. While higher batch repetitions did help to some extent, the additional computational cost made it impractical.

Given the surprising result that an ensemble size of $M = 2$ provides no performance improvement over the standard V-MoE and that increasing M further provides worse performance, there seems to be some incompatibility between MIMO and V-MoE. In fact, our investigations revealed that ViT is the source of the problems since applying MIMO to vanilla ViT without experts resulted in the same trends as for V-MoE. Thus, we hypothesise that the differences between ViT and ResNet—the architecture to which MIMO was originally applied by Havasi et al. (2020)—are responsible for MIMO's poor performance when applied to ViT.

Difference 1: Class token. One of the differences between ViT and ResNet is that ViT makes use of a special learnable class token to classify an image (see Dosovitskiy et al. (2021) for details). ResNet, on the other hand, makes use of the representation from an entire image for classification. We tried two strategies to mitigate this difference:

1. We applied the global average pooling (GAP) and multi-head attention pooling (MAP) classification strategies introduced in Dosovitskiy et al. (2021) and Zhai et al. (2022), respectively. In short, both of these methods make use of all the tokens from an image for classification. However, neither of these strategies made a significant difference to the relative performance of MIMO and ViT. In fact, the choice of classification method was the least impactful hyperparameter in our sweep.

2. Rather than learning a single class token, we learnt M class tokens. This strategy resulted in MIMO with $M = 2$ outperforming ViT. However, for $M > 2$ the improvement was small enough that ViT still outperformed MIMO.

Difference 2: Attention. The other major difference between ViT and ResNet is the building block for each model. While ResNets are primarily composed of convolution operations, ViT makes heavy use of attention. We hypothesised that attention is less suited to separating the information for M input images stored in the channel dimension of a single image. We tried two strategies to mitigate this potential issue:

1. We applied the hybrid architecture, described in Dosovitskiy et al. (2021), in which the input sequence to ViT is formed by CNN feature maps. We used ResNet-14 and ResNet-50. In both cases, we found that the strategy boosted the performance of ViT and MIMO equally.

2. Rather than concatenating images in the channel dimension, we concatenated them in the width dimension, resulting in 3 times as many patches for ViT to process.

This strategy was successful in the sense that the MIMO performance for $M > 2$ improved significantly. However, the significant additional computational cost made it an infeasible solution.

Our findings suggest that MIMO and ViT are indeed somewhat incompatible. Unfortunately, none of our proposed solutions to this problem provided high enough predictive performance increases (or indeed low enough computational cost increases in some cases) to warrant immediate further investigation.

C.8 Additional Experimental Results

In this section, we expand on various experiments presented in Sections 5.3 to 5.5. In experiments considering multiple ViT families, we also include B/16, which was excluded from the main text for clarity.

C.8.1 Static versus Adaptive Combination

Here we continue the investigation into static versus adaptive combination from Section 5.3.

Individual gains with respect to E, K and M. Figure C.2 shows the effect of increasing the various 'ensemble size' parameters for a deep ensemble of V-MoEs. In particular, we investigate the static combination axis M (the number of ensemble members), as well as the two adaptive axes—K (the number of experts chosen per patch) and E (the total number of experts).

When investigating the effect of K, we fix $E = 32$ and average over $M \in \{1, .., 8\}$. Similarly, when investigating M, we fix $E = 32$ and average over $K \in \{1, .., 8\}$. When investigating the effect of E we fix $K = 2$ and average over $M \in \{1, .., 8\}$. As a result of this procedure, the exact values of the curves are not directly comparable. However, we can still examine the relative trends.

Specifically, we note that while the variation in K and M curves is roughly of the same size, the variation in the E curve is smaller. We also note that there is very little variation beyond $E = 8$ (note the difference in the scales of the axes for the curves). These observations motivate the design of E^3, where we split the sub-models along the E axis, in order to better take advantage of the experts.

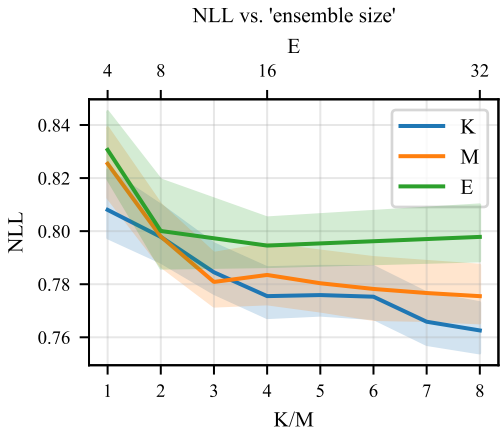


Figure C.2: Comparison for the impact on ImageNet NLL of variations in K, E and M. The underlying model is ViT-S/32.

C.8.2 An Additional Motivating Experiment – Deep Ensembles of V-MoE with Fewer Experts

As an additional motivation for combining sparse MoEs and ensembles, Table C.6 compares the performance of a V-MoE with $E = 32$ total experts and ensembles of V-MoEs with ($M = 2$, $E = 16$) and ($M = 4$, $E = 8$), for both $K = 1$ and $K = 2$. We see that, in terms of NLL, ($M = 4$, $E = 8$) is better than ($M = 2$, $E = 16$) which is in turn better than ($M = 1$, $E = 32$). We see similar results for Error and ECE.

We note that this result is especially remarkable since the *individual* upstream (and later, downstream) models are such that $E = 8$ is worse than $E = 16$, which in turn performs worse than $E = 32$; ensembling thus manages to counterbalance the poorer individual model performance. This suggests that the efficient ensembling—i.e., the combination of multiple models within a single model—of sparse MoEs could lead to strong performance while reducing computational costs.

Table C.6: Comparison of upstream deep ensembles of V-MoE-B/32 models with fewer experts.

	K	M	E	NLL ↓	Error ↓	ECE ↓	KL ↑
V-MoE	1	1	32	0.642 ± 0.002	16.90 ± 0.05	0.029 ± 0.001	–
		2	16	0.588	15.97	0.015	0.211
		4	8	0.577	15.82	0.017	0.228
	2	1	32	0.638 ± 0.001	16.76 ± 0.05	0.033 ± 0.001	–
		2	16	0.583	15.75	0.016	0.211
		4	8	0.580	15.94	0.015	0.184

C.8.3 The Roles of Ensemble Diversity and Individual Model Performance

Table C.7 shows the individual member performance for each of our efficient ensemble variants as well as upstream and downstream deep ensembles, for sizes $M = 2$ and $M = 4$. For each method and ensemble size, the first row shows the combined ensemble performance, and the following rows show the performance of the individual ensemble members.

For E^3, the gap between single members and their ensemble is considerable. This is reminiscent of deep ensembles (both variants) and in stark contrast with what we observe for the other efficient ensembles: BE ViT and MIMO V-MoE. The diversity of E^3 is comparable to that of upstream deep ensembles and much larger than downstream deep ensembles. For example, if we compare MIMO V-MoE (M=2) and E^3 (M=2), the single members of E^3 are all worse in NLL, ACC, and ECE than the single members of MIMO. However, the performance gap between the individual members and the ensemble is much larger for E^3 than MIMO (where there is almost no difference). The diversity of the individual models in E^3 is key to its strong performance. Thus, E^3 outperforms MIMO.

In short, this suggests that E^3 approximates a deep ensemble of smaller V-MoE models. That is, with E^3, we are able to take advantage of the overparameterization of the experts to create a rich set of ensemble members within a single model. Each ensemble member has a large number of non-shared parameters, thus high induced diversity. In comparison, BE only has a few vectors specific to each member.

Table C.7: Comparison of the individual ensemble member performance and combined ensemble performance for BE ViT, MIMO V-MoE ($K = 2$, BR=1), E^3 ($K = 1$), as well as upstream and downstream V-MoE ($K = 1$) ensembles.

			NLL ↓	Error ↓	ECE ↓	KL ↑
BE ViT	M=2	ensemble	0.682 ± 0.003	18.47 ± 0.05	0.021 ± 0.000	0.040 ± 0.001
		member 0	0.693 ± 0.003	18.68 ± 0.05	0.025 ± 0.000	—
		member 1	0.693 ± 0.003	18.68 ± 0.04	0.025 ± 0.001	—
	M=4	ensemble	0.675 ± 0.003	18.40 ± 0.09	0.017 ± 0.000	0.035 ± 0.001
		member 0	0.690 ± 0.003	18.70 ± 0.09	0.022 ± 0.000	—
		member 1	0.690 ± 0.003	18.70 ± 0.08	0.023 ± 0.000	—
		member 2	0.690 ± 0.003	18.70 ± 0.07	0.023 ± 0.000	—
		member 3	0.691 ± 0.003	18.70 ± 0.07	0.023 ± 0.000	—
MIMO V-MoE	M=2	ensemble	0.636 ± 0.002	16.97 ± 0.04	0.028 ± 0.001	0.001 ± 0.000
		member 0	0.636 ± 0.002	16.97 ± 0.04	0.028 ± 0.001	—
		member 1	0.636 ± 0.002	16.97 ± 0.04	0.028 ± 0.000	—
	M=4	ensemble	0.672 ± 0.001	17.72 ± 0.04	0.037 ± 0.000	0.001 ± 0.000
		member 0	0.672 ± 0.001	17.74 ± 0.05	0.037 ± 0.000	—
		member 1	0.672 ± 0.001	17.71 ± 0.05	0.037 ± 0.000	—
		member 2	0.672 ± 0.001	17.72 ± 0.05	0.037 ± 0.000	—
		member 3	0.673 ± 0.001	17.73 ± 0.05	0.037 ± 0.000	—
E^3	M=2	ensemble	0.622 ± 0.001	16.70 ± 0.03	0.018 ± 0.000	0.217 ± 0.003
		member 0	0.671± 0.003	17.74 ± 0.06	0.038 ± 0.001	—
		member 1	0.683 ± 0.002	17.94 ± 0.005	0.038 ± 0.001	—
	M=4	ensemble	0.624 ± 0.001	16.99 ± 0.03	0.013 ± 0.000	0.164 ± 0.001
		member 0	0.677 ± 0.002	18.04 ± 0.05	0.034 ± 0.001	—
		member 1	0.685 ± 0.001	18.23 ± 0.05	0.034 ± 0.001	—
		member 2	0.691 ± 0.002	18.35 ± 0.07	0.035 ± 0.001	—
		member 3	0.697 ± 0.002	18.47 ± 0.08	0.035 ± 0.001	—
Up-DE	M=2	ensemble	0.588 ± 0.001	15.74 ± 0.05	0.017 ± 0.001	0.214 ± 0.001
		member 0	0.640 ± 0.001	16.82 ± 0.04	0.030 ± 0.000	—
		member 1	0.645 ± 0.002	16.97 ± 0.06	0.029 ± 0.002	—
	M=4	ensemble	0.561 ± 0.001	15.10 ± 0.03	0.020 ± 0.000	0.214 ± 0.001
		member 0	0.639 ± 0.001	16.80 ± 0.03	0.030 ± 0.000	—
		member 1	0.641 ± 0.001	16.84 ± 0.05	0.030 ± 0.000	—
		member 2	0.642 ± 0.001	16.90 ± 0.03	0.031 ± 0.000	—
		member 3	0.647 ± 0.002	17.05 ± 0.06	0.028 ± 0.002	—
Down-DE	M=2	ensemble	0.620 ± 0.001	16.44 ± 0.04	0.023 ± 0.000	0.073 ± 0.001
		member 0	0.642 ± 0.001	16.83 ± 0.03	0.030 ± 0.000	—
		member 1	0.643 ± 0.001	16.84 ± 0.03	0.030 ± 0.000	—
	M=4	ensemble	0.607 ± 0.000	16.17 ± 0.02	0.021 ± 0.001	0.073 ± 0.000
		member 0	0.641 ± 0.001	16.82 ± 0.03	0.030 ± 0.000	—
		member 1	0.642 ± 0.001	16.85 ± 0.03	0.030 ± 0.000	—
		member 2	0.643 ± 0.001	16.89 ± 0.04	0.031 ± 0.000	—
		member 3	0.643 ± 0.001	16.93 ± 0.07	0.031 ± 0.000	—

C.8.4 Extended Results for Few-shot Learning

In Figure C.3, we extend the few-shot learning results of Figure 5.4 to also include 1, 5, and 25-shot. Additionally, we show results for the weighted aggregation strategy mentioned in Section C.1.5.

We confirm the result that the few-shot performance for E^3 gets better, relative to the other baselines, with larger ViT families. Additionally, we see that E^3 performance seems to get better, again relative to the other baselines, with more shots. This phenomenon can most easily be noticed by comparing the results for S/32 across different numbers of shots. Finally, we see that the trends with and without the weighted mean are the same.

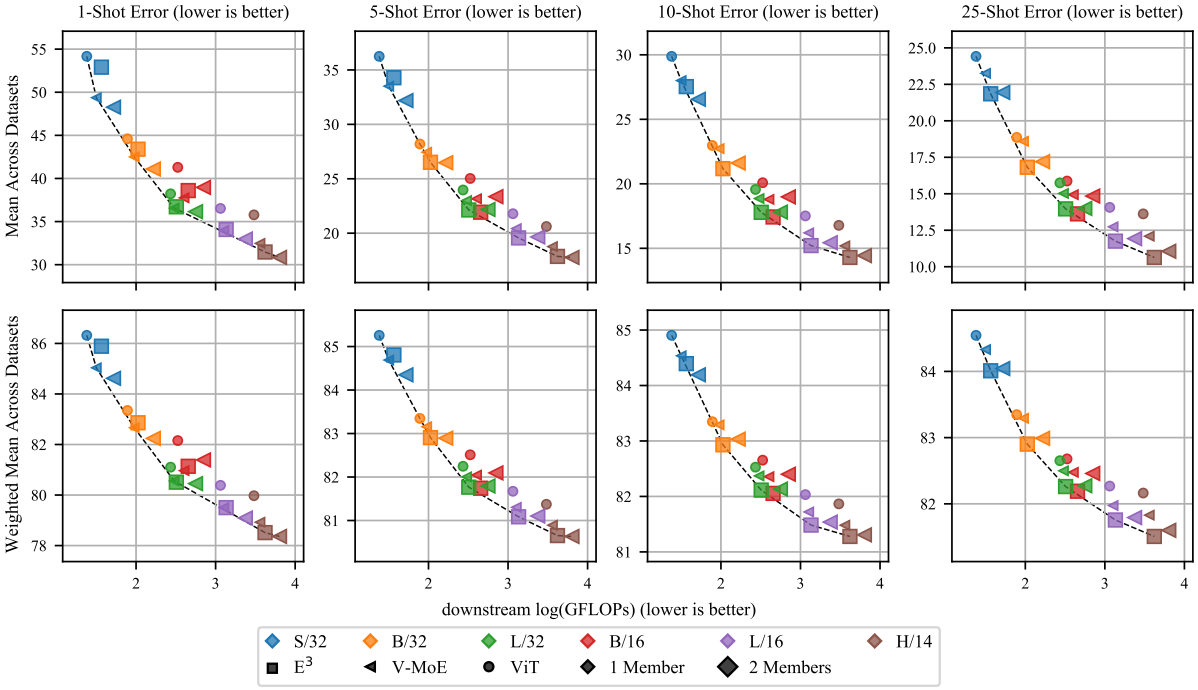


Figure C.3: Extended few-shot results from Figure 5.4 with an additional aggregation method and numbers of shots.

C.8.5 Extended Results for OOD Detection

Here we extended the OOD results of Figure 5.6. Specifically, we add CIFAR100 as an in-distribution dataset and Describable Textures Dataset (DTD) (Cimpoi et al., 2014) as an OOD dataset. We also add the area under the receiver operating characteristic (AUC (ROC)) and the area under the precision-recall curve (AUC (PR)) as metrics. Figures C.4 and C.5 contain the results with CIFAR10 and CIFAR100 as the in-distribution datasets, respectively.

As in Figure 5.6, we see that E^3 performs better (relative to the other baselines) for larger ViT families. Furthermore, E^3 seems to perform better in near OOD detection (i.e., CIFAR10 versus CIFAR100, and vice versa) than far Out-of-distribution (OOD) detection. Finally, we see that these trends are consistent across OOD metrics.

C.8.6 Extended Results for ImageNet

In this section, we extend the results for ImageNet and the corrupted variants presented in Figures 5.4, 5.5 and 5.7. In addition to NLL, classification error, ECE (for standard ImageNet), and Brier score, Figure C.6 provides classification error and ECE for all ImageNet variants.

Most of the trends observed in Section 5.5 remain true:

- E^3 tends to be Pareto efficient in the presence of a distribution shift.
- For smaller ViT families, V-MoE outperforms ViT in the presence of distribution shift.
- E^3 improves ECE over ViT and V-MoE.
- E^3 improves classification performance.
- ViT consistently provides better ECE than V-MoE.

However, there are some exceptions:

- **ImageNet-A classification error.** All models (including E^3) under-perform relative to ViT-S/32 and ViT-H/14.
- **ECE for ImageNet-C, ImageNet-A, and ImageNet-V2.** Interestingly, for the non-standard ImageNet variants, and in particular for ImageNet-A, there is a strong correlation between lower ECE and larger ViT families.

We also find that the results for classification error and Brier score follow those for NLL closely.

C.8.7 Additional CIFAR10, CIFAR100, Flowers, and Pets Results

Here we extend the results for ImageNet and the corrupted variants presented in Figures 5.4, 5.5 and 5.7 to four additional datasets. Figures C.7, C.8, C.9 and C.10 provide

results for CIFAR10, CIFAR100, Oxford Flowers 102, and Oxford IIIT Pet, respectively. As in Section C.8.6, we find that the results are similar to those in Section 5.5.

Compared to ImageNet, for CIFAR10, CIFAR10-C, and CIFAR100, E^3 seems to perform even better relative to the other baselines. Note, for example, that E^3 is Pareto efficient (even for S/32) in the cases of CIFAR10-C and CIFAR100 NLL. As in Section C.8.6, we see that the ECE has a stronger downward trend with respect to increased ViT family size for shifted test data.

For Flowers and Pets, where we only have results for smaller ViT families, E^3 seems to underperform. However, the performance for L/32 is better than for S/32 and B/32, which suggests that the results for these datasets are consistent with the other datasets presented in this work and, therefore, that we should expect E^3's predictive performance to keep improving with larger models.

C.8.8 Efficient Ensemble of Experts and V-MoE with larger values of K and M

Figure C.11 and Figure C.12 show the effect of varying K on E^3 and V-MoE, and the effect of varying M on E^3, respectively. We make the following observations:

- In almost all cases, increasing K or M does not result in Pareto efficient models.
- For V-MoE, increasing K seems to help in most cases, except for ECE performance, where it usually hurts.
- For E^3, going from $K = 1$ to $K = 2$ seems to help in most cases but going from $K = 2$ to $K = 4$ usually hurts. Going from $K = 1$ to $K = 4$ still helps, but to a lesser extent than from $K = 1$ to $K = 2$.
- For E^3, increasing M either doesn't make a consistent and significant difference or hurts (e.g. in OOD detection).

These conclusions should, however, be considered with caution. Recall that the upstream checkpoints used for fine-tuning all V-MoE and E^3 models in this work are V-MoE models with $K = 2$. Thus, the results in this experiment are confounded by upstream and downstream checkpoint mismatch for all E^3 models and all V-MoE models with $K \neq 2$. This phenomenon was observed in Riquelme et al. (2021) for V-MoE models. I.e., performance of downstream V-MoE models with mismatched values of K was relatively worse than those with matched values of K; see their Appendix E.4 (Figures 33-35). We also hypothesise that it is more difficult to train downstream E^3

models with larger values of M from upstream V-MoE models because in each subset of the experts some common expert specialisations will need to be duplicated. Our ensemble members are fine-tuned from an upstream V-MoE with a predefined total number of experts ($E = 32$), meaning that increasing M decreases the number of experts available to each ensemble member (with E/M experts per member). This could also impact the performance of E^3 with larger sizes of M.

Upstream vs. Downstream Mismatch

Table C.8: ImageNet performance of E^3 models fine-tuned from V-MoE-B/32 checkpoints with $K = 2$ or $K = 4$, and $E = 32$. $K_{\text{upstream}} = 2$ results are averaged over 8 random seeds, while $K_{\text{upstream}} = 4$ results are averaged over 3 seeds.

	K	M	K_{upstream}	NLL ↓	Error ↓
E^3	1	2	2	**0.622** ± 0.001	**16.70** ± 0.03
			4	**0.622** ± 0.001	16.81 ± 0.05
	1	4	2	0.624 ± 0.001	16.99 ± 0.03
			4	**0.622** ± 0.000	**16.93** ± 0.01

Here we investigate whether upstream versus downstream mismatch partially explains the counter-intuitive result that increasing M can result in worse performance for E^3. We train downstream E^3 models with ($K = 1$, $M = 2$) and ($K = 1$, $M = 4$) from upstream V-MoE checkpoints with $K = 2$ and $K = 4$. Table C.8 shows the results. We see that ($K = 1$, $M = 2$) does not seem to benefit from increasing K_{upstream} from 2 to 4, despite the fact that the upstream $K = 4$ model is better than the upstream $K = 2$ model (NLL upstream is 8.18 for $K = 4$ and 8.27 for $K = 2$). On the other hand, ($K = 1$, $M = 4$) does benefit from having K=4 upstream.

This confirms that the mismatch between upstream and downstream models is one of the factors explaining the results of E^3 for growing M. However, we also see that the best performing model is ($K_{\text{upstream}} = 2$, $K = 1$, $M = 2$), which suggests that there are other confounding factors, such as those mentioned above.

C.8.9 Extended Results for the Tiling with Increasing Parameter Sharing Ablation

In Table C.9, we extend the results for our parameter sharing ablation in Section 5.4.2, from $K = 2$ to $K = 1$. We see that the results remain the same in this case.

Table C.9: Extension of Table 5.3, showing the impact of parameter sharing in E^3, for $K = 1$.

Overlap	NLL ↓	Error ↓	ECE ↓	KL ↑
0 (=E^3)	**0.622** ± 0.001	**16.70** ± 0.03	**0.018** ± 0.000	**0.217** ± 0.003
2	0.627 ± 0.003	16.83 ± 0.07	0.022 ± 0.001	0.194 ± 0.005
4	0.634 ± 0.002	16.92 ± 0.07	0.024 ± 0.001	0.178 ± 0.004
8	0.642 ± 0.001	17.04 ± 0.10	0.028 ± 0.001	0.151 ± 0.009
16	0.659 ± 0.004	17.28 ± 0.12	0.036 ± 0.001	0.103 ± 0.009

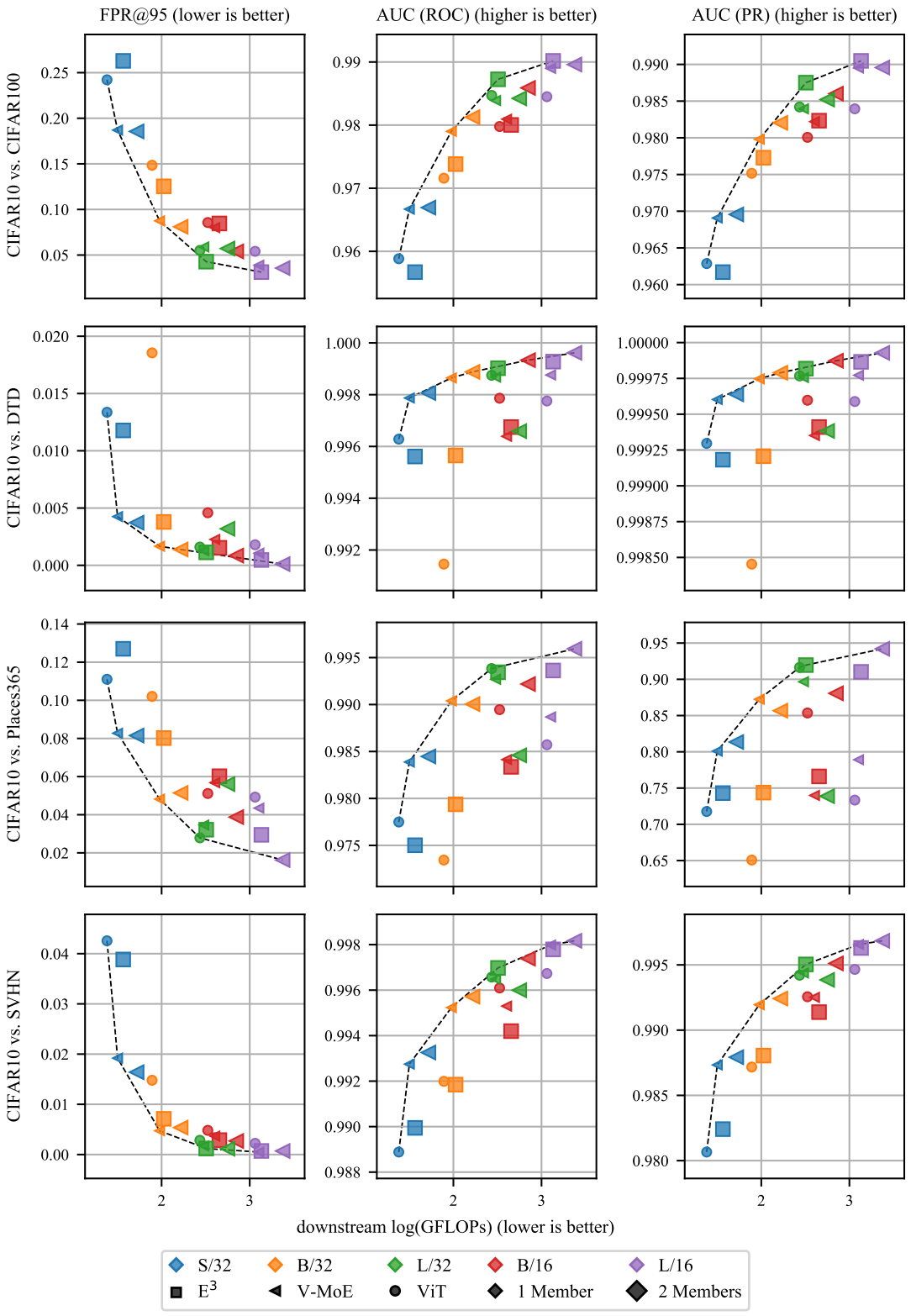


Figure C.4: Extended OOD detection the results from Figure 5.6 with an additional OOD dataset and more metrics.

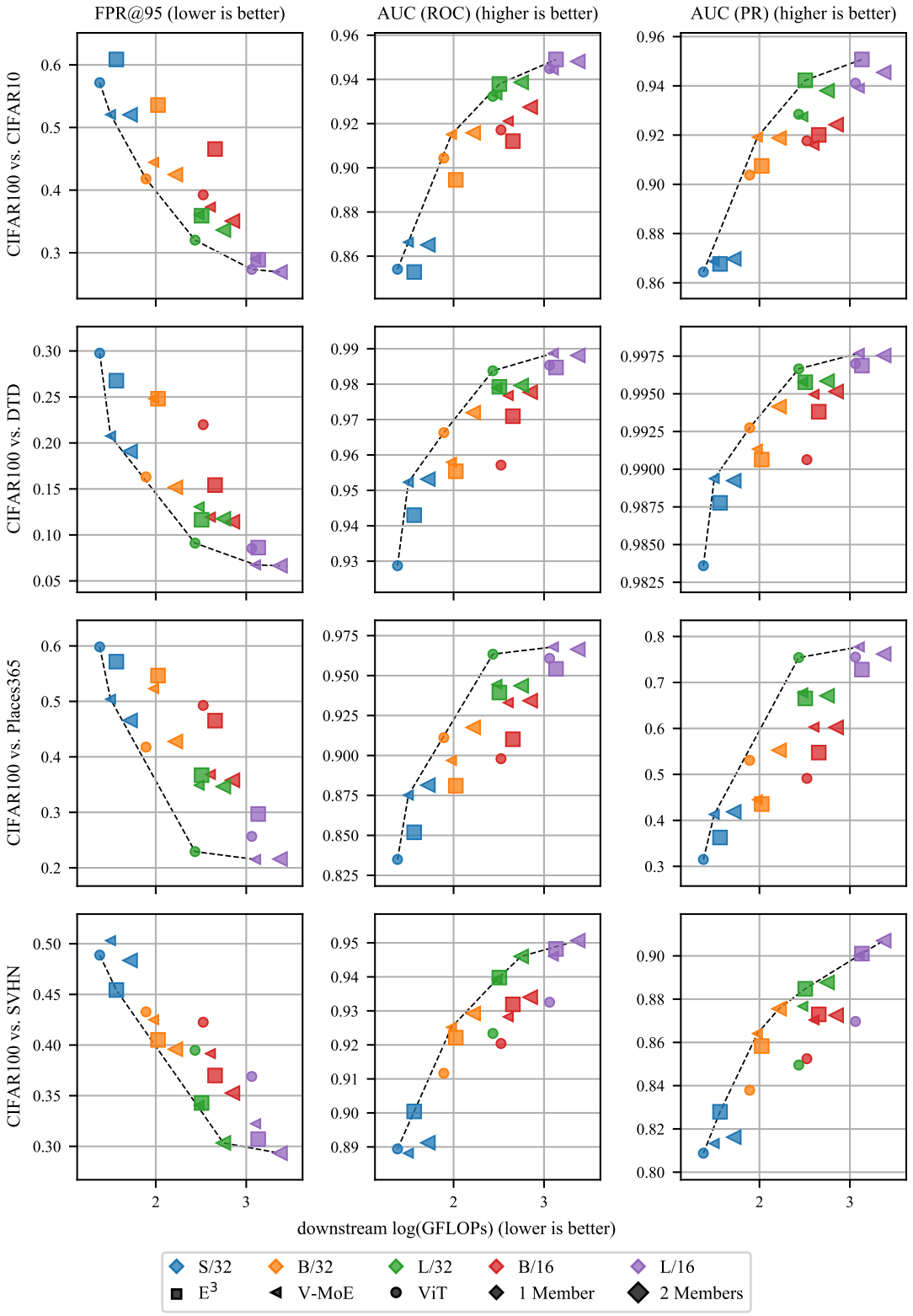


Figure C.5: Extended OOD detection results from Figure 5.6 with CIFAR100 as the in-distribution dataset, an additional OOD dataset, and more metrics.

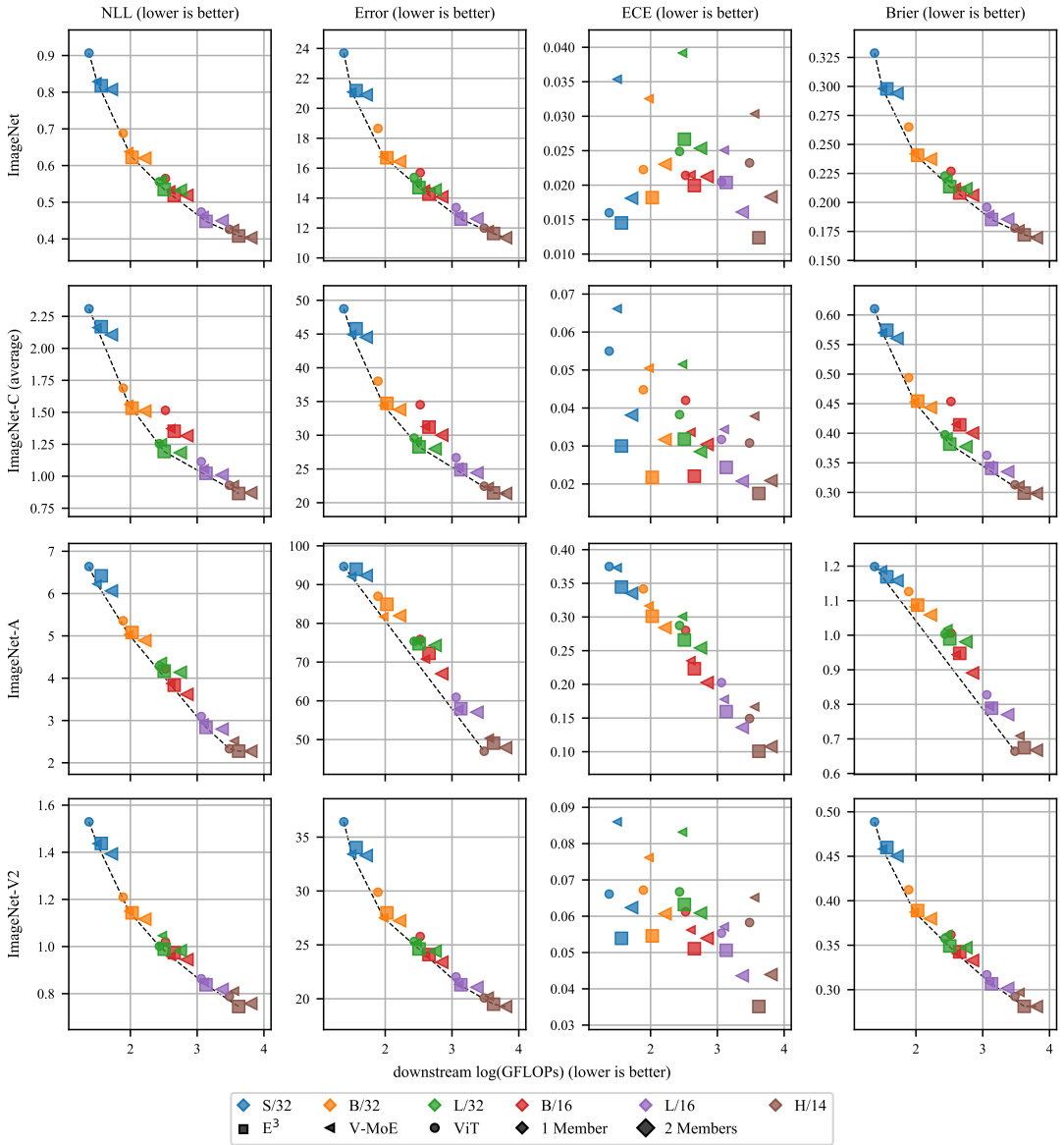


Figure C.6: Extended results from Figures 5.4, 5.5 and 5.7 with additional metrics.

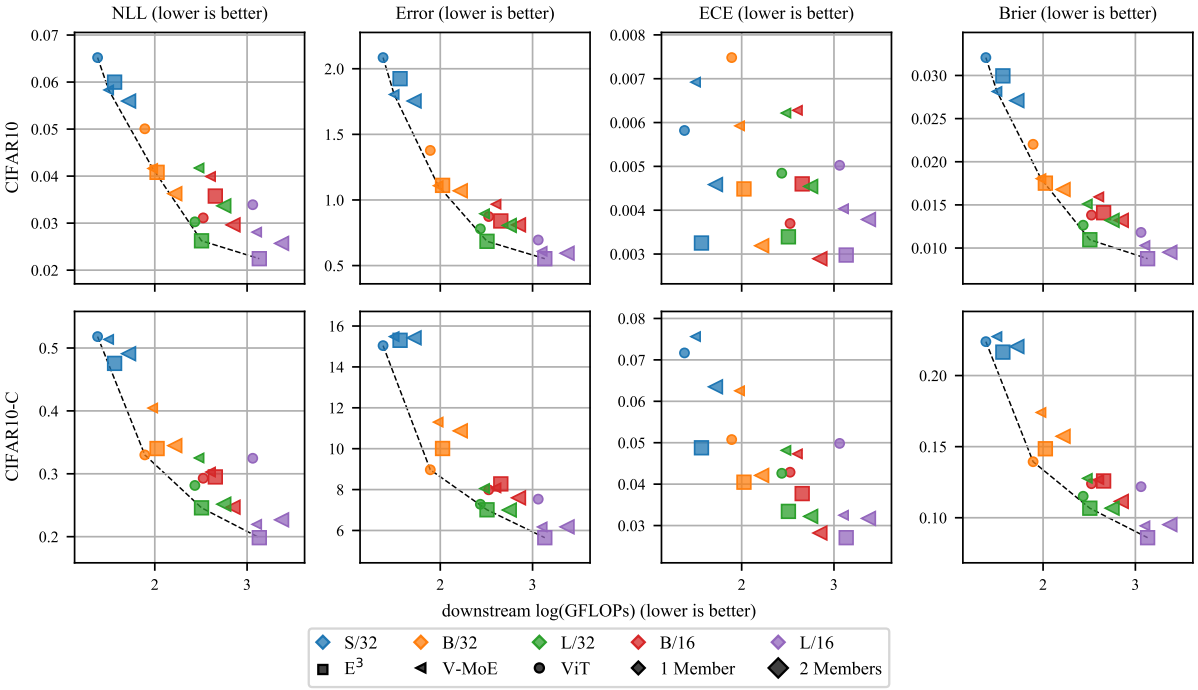


Figure C.7: Results for CIFAR10 and CIFAR10-C.

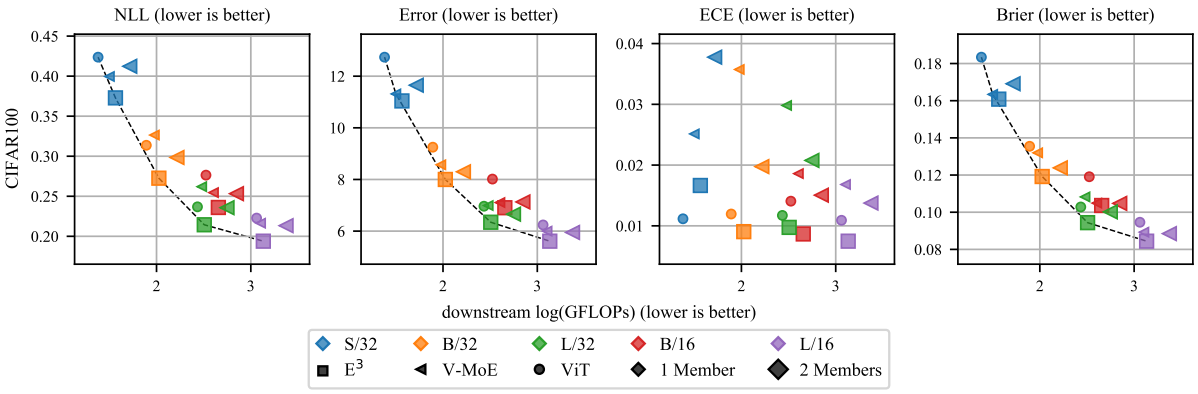


Figure C.8: Results for CIFAR100.

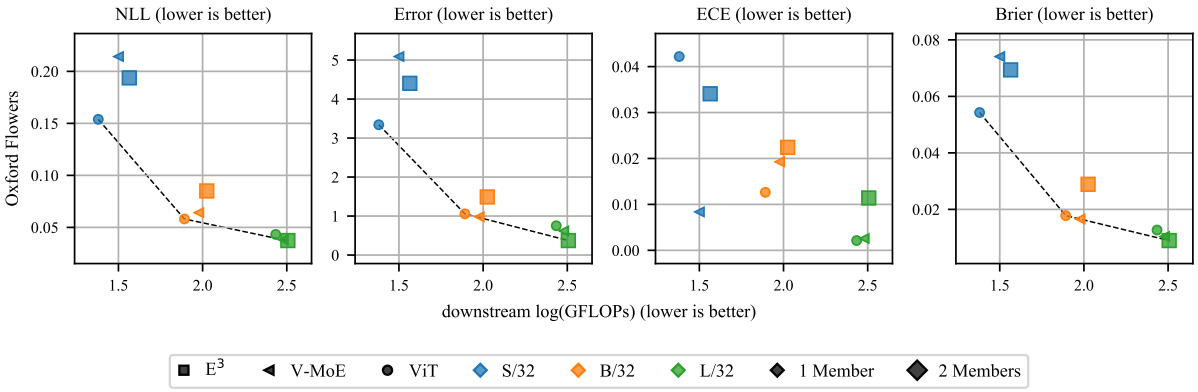


Figure C.9: Results for Oxford Flowers 102.

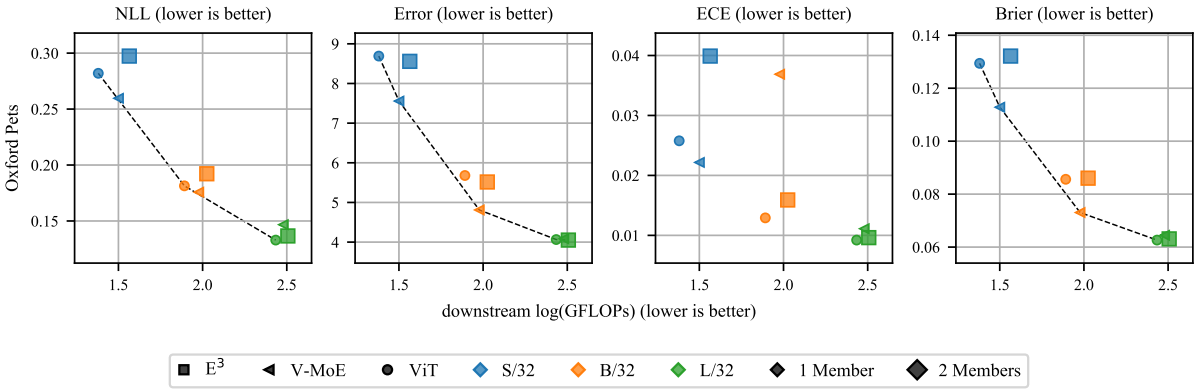


Figure C.10: Results for Oxford IIIT Pet.

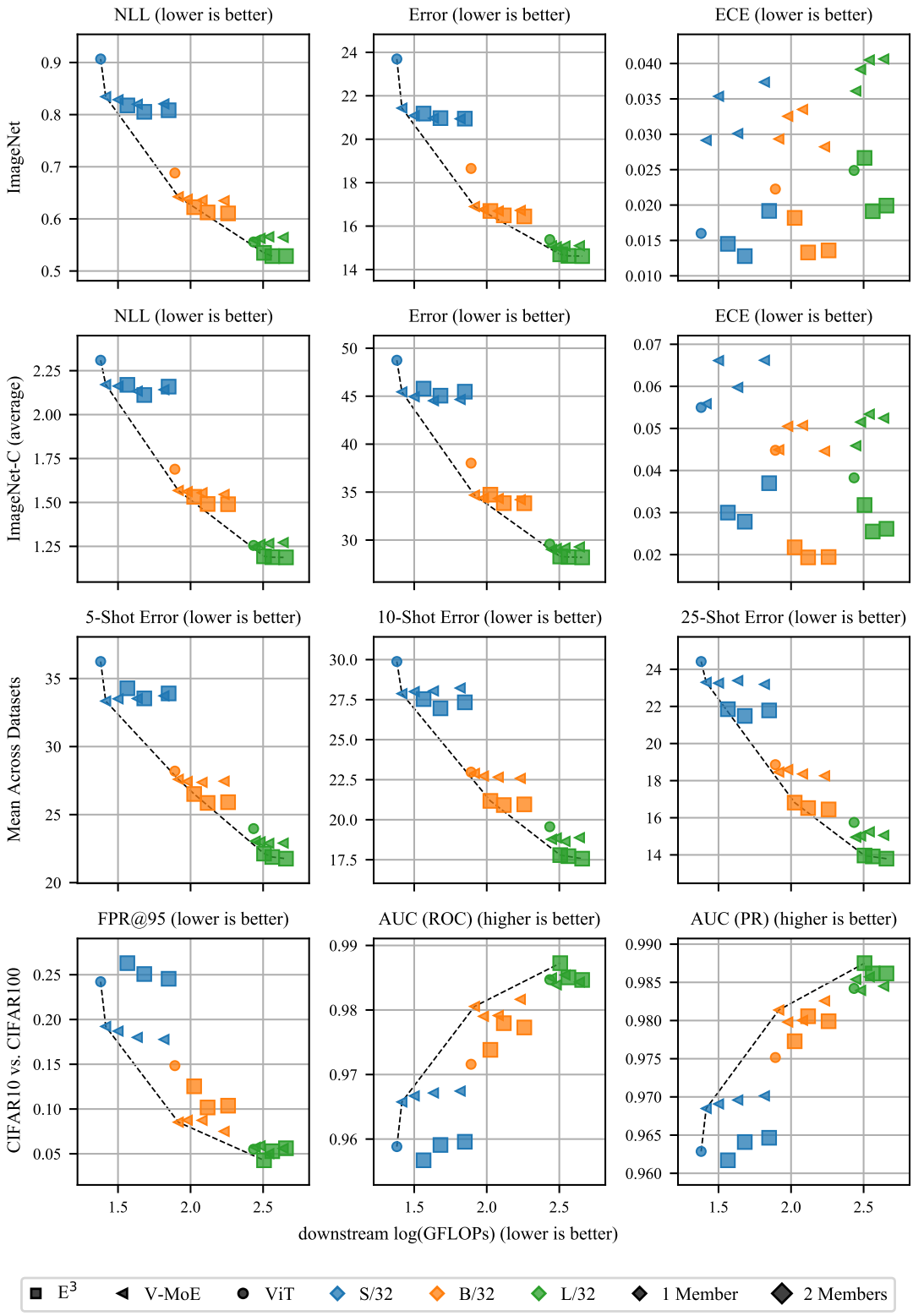


Figure C.11: Results for V-MoE with $K \in \{1, 2, 4, 8\}$ and E^3 with $K \in \{1, 2, 4\}$. Models with larger values of K have larger FLOPs.

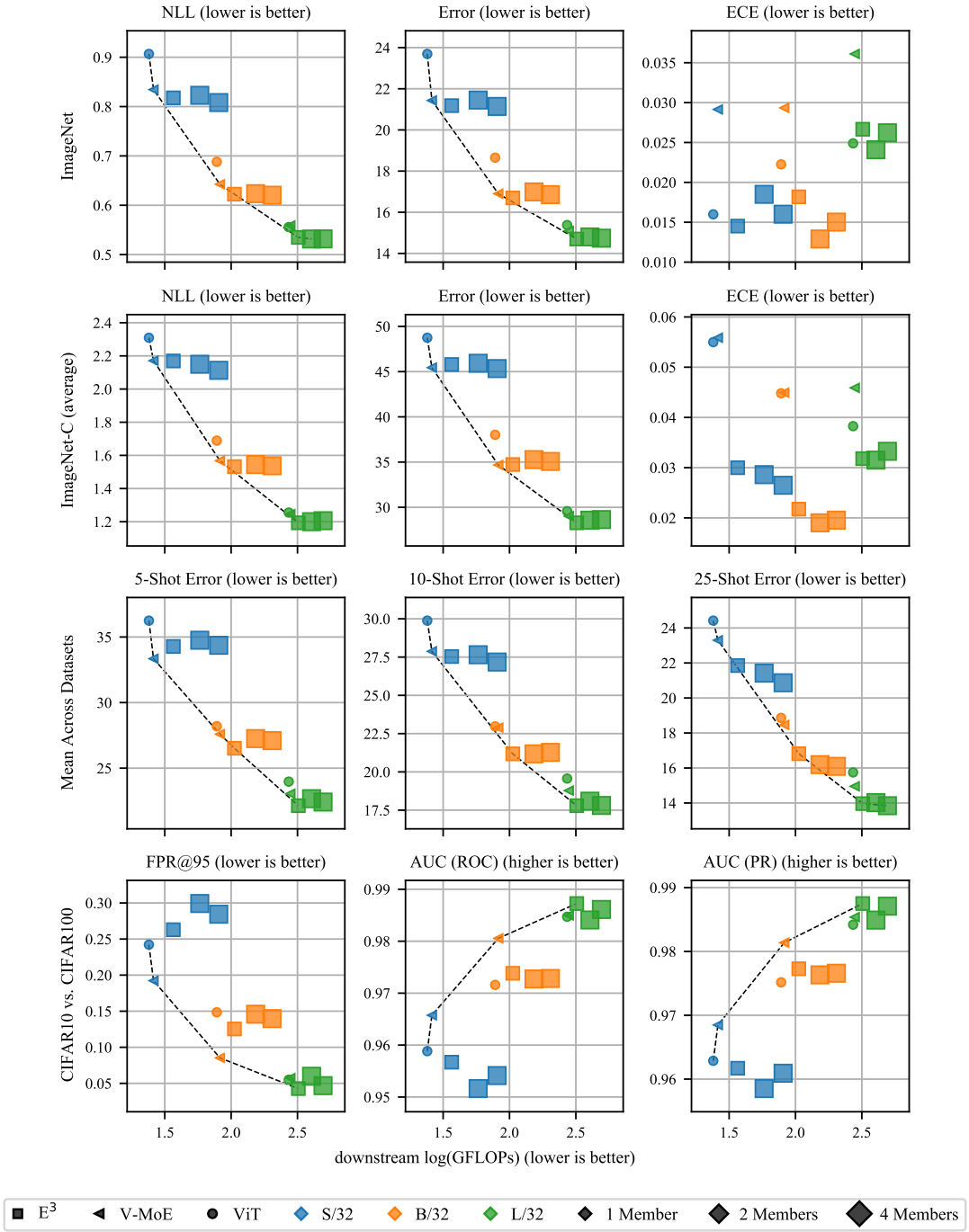


Figure C.12: Results for E^3 with $M = 4$ and $K \in \{1, 2\}$.

C.8.10 Summary for NLL under Distribution Shift

Table C.10 shows the percentage improvement for E^3 versus V-MoE in NLL (i.e., $\frac{\text{NLL}_{\text{V-MoE}} - \text{NLL}_{\text{E}^3}}{\text{NLL}_{\text{V-MoE}}} \times 100$, with positive values thus indicating that E^3 improves upon V-MoE) averaged over ImageNet-C, ImageNet-A, and ImageNet-V2, for a given ViT family and (K, M) in $\{(1, 2), (2, 2)\}$ (compared to V-MoE with $K = 2$ and $K = 4$, respectively). We see that for all ViT families except S/32, E^3 outperforms V-MoE. This result is not unexpected, since ensembles tend to provide improved performance under distribution shift relative to single models (Lakshminarayanan et al., 2017; Ovadia et al., 2019; Havasi et al., 2020).

Table C.10: Percentage improvement for E^3 vs V-MoE in NLL under distribution shift, averaged over ImageNet-C, ImageNet-A, and ImageNet-V2.

	S/32	B/32	B/16	L/32	L/16	H/14
E^3 $(K = 1, M = 2)$ vs. V-MoE $(K = 2)$	-1.15	0.52	0.31	5.34	2.78	8.33
E^3 $(K = 2, M = 2)$ vs. V-MoE $(K = 4)$	-0.21	3.91	1.33	7.37	2.27	—

C.8.11 Preliminary ImageNet Results without Pre-training

Training large-scale sparse MoEs on datasets such as ImageNet and in absence of any pre-training is a difficult task. Indeed, the massive number of parameters causes models to severely overfit in that regime. In practice, we need to combine various regularisation techniques to control this overfitting.

To obtain preliminary results of E^3 trained from scratch on ImageNet, we adapt the recipe proposed in the code released by Riquelme et al. (2021). The recipe is itself based on the regularisation protocol of Steiner et al. (2022) ("AugReg"), which trained performant dense ViT models from scratch on ImageNet, without pre-training. Overall, compared to the default training schema used for this paper, we add:

- Mixup (Zhang et al., 2018),
- Weight decay,
- Dropout (Srivastava et al., 2014) on expert MLPs.

The baseline configuration for V-MoE was tuned according to the hyperparameter search space defined by Steiner et al. (2022), with an extra sweep over expert MLP dropout in $\{0.1, 0.2\}$. Selecting according to validation accuracy, the optimal V-MoE setting

Table C.11: ImageNet performance (means ± standard errors over 3 seeds) of V-MoE and E^3 for a S/32 backbone, without pre-training.

	K	NLL ↓	Error ↓
E^3 ($M = 2$)	1	**1.420** ± 0.007	**23.78** ± 0.22
V-MoE	2	1.478 ± 0.003	24.45 ± 0.08

was `medium2`, i.e., mixup ratio 0.5 and RandAugment (Cubuk et al., 2020) parameters 2 and 15 (2 augmentations applied of magnitude 15), alongside expert dropout 0.2. For E^3, these regularisation-related hyperparameters were kept fixed and were not further tuned. More precisely, we keep `medium2` and tune only the learning rate in $\{0.001, 0.003\}$; we thus suspect we could improve our current results for E^3 by considering a broader sweep of hyperparameters (e.g., re-tuning all the regularisation-related hyperparameters).

In Table C.11, we report the performance for both V-MoE and E^3 with a S/32 backbone. In terms of both NLL and classification error, E^3 outperforms V-MoE. The model checkpoint, along with our code, can be found at `https://github.com/google-research/vmoe`.

C.8.12 Upstream & Downstream versus Downstream-only Ensembles

In Section 5.5, and Section C.8 include *downstream* deep ensembles (down-DE) of V-MoE, and in some cases ViT, as a baseline. This choice was motivated by the fact that like ViT, V-MoE, and E^3, down-DE requires only a single upstream checkpoint, which all of the methods more comparable. However, it is clear that using different upstream checkpoints and then further fine-tuning each of these with different random seeds to construct an *upstream* deep ensemble (up-DE) would result in more varied ensemble members and as a result, a better performing ensemble. This idea has recently been explored by Mustafa et al. (2020).

Thus, for completeness, we also investigate the effects of upstream ensembling on V-MoE. Table C.12 compares the performance of upstream and downstream V-MoE ($K = 1$) ensembles of sizes $M = 2$ and $M = 4$. Across the range of metrics, for both ImageNet and ImageNet-C, for all ViT families, and for both values of M, we see that up-DE outperforms down-DE. In fact, up-DE with $M = 2$ is very often better than or equal to down-DE with $M = 4$. This is especially true for the diversity metrics, which indicates that diversity is indeed the driver for improved performance in up-DE. Not

shown in the table is the very large computational cost associated with training upstream ensembles.

Table C.12: Comparison of upstream and downstream ensembles of V-MoE with ($K = 1$).

			ImageNet						ImageNet-C					
		M	NLL ↓	Error ↓	ECE ↓	KL ↑	Cos. Sim. ↓	Norm. Dis. ↑	NLL ↓	Error ↓	ECE ↓	KL ↑	Cos. Sim. ↓	Norm. Dis. ↑
H/14	down-DE	2	0.403 ± 0.000	11.35 ± 0.05	0.018 ± 0.001	0.079 ± 0.003	0.974 ± 0.001	0.488 ± 0.006	0.871 ± 0.012	21.37 ± 0.20	**0.021** ± 0.001	0.218 ± 0.002	0.925 ± 0.001	0.628 ± 0.003
	up-DE	2	0.391 ± 0.000	11.12 ± 0.10	0.016 ± 0.001	**0.126** ± 0.008	**0.963** ± 0.002	0.625 ± 0.012	0.839 ± 0.011	20.66 ± 0.22	**0.022** ± 0.000	**0.355** ± 0.007	**0.892** ± 0.002	0.809 ± 0.006
	down-DE	4	0.392 ± 0.000	11.20 ± 0.000	0.014 ± 0.000	0.083 ± 0.000	0.973 ± 0.000	0.509 ± 0.000	0.851 ± 0.000	20.97 ± 0.000	**0.021** ± 0.000	0.221 ± 0.000	0.923 ± 0.000	0.650 ± 0.000
	up-DE	4	**0.375** ± 0.000	**10.66** ± 0.000	**0.013** ± 0.000	**0.129** ± 0.000	**0.963** ± 0.000	**0.652** ± 0.000	**0.792** ± 0.000	**19.61** ± 0.000	0.032 ± 0.000	**0.361** ± 0.000	**0.892** ± 0.000	**0.850** ± 0.000
L/16	down-DE	2	0.450 ± 0.002	12.62 ± 0.04	0.016 ± 0.000	0.061 ± 0.001	0.979 ± 0.000	0.419 ± 0.002	1.010 ± 0.006	24.43 ± 0.12	**0.021** ± 0.000	0.168 ± 0.002	0.936 ± 0.001	0.539 ± 0.003
	up-DE	2	0.434 ± 0.000	12.23 ± 0.04	**0.014** ± 0.000	**0.118** ± 0.001	**0.964** ± 0.000	0.584 ± 0.001	0.961 ± 0.001	23.46 ± 0.03	0.023 ± 0.000	**0.342** ± 0.001	**0.890** ± 0.000	0.766 ± 0.001
	down-DE	4	0.440 ± 0.002	12.39 ± 0.06	0.015 ± 0.000	0.061 ± 0.001	0.979 ± 0.000	0.425 ± 0.002	0.983 ± 0.006	23.95 ± 0.12	**0.020** ± 0.000	0.166 ± 0.001	0.937 ± 0.000	0.547 ± 0.002
	up-DE	4	**0.418** ± 0.000	**11.86** ± 0.01	**0.013** ± 0.000	**0.118** ± 0.000	**0.964** ± 0.000	**0.603** ± 0.001	**0.916** ± 0.001	**22.45** ± 0.02	0.034 ± 0.000	**0.341** ± 0.000	**0.890** ± 0.000	**0.800** ± 0.001
L/32	down-DE	2	0.533 ± 0.002	14.55 ± 0.04	0.025 ± 0.001	0.092 ± 0.001	0.969 ± 0.000	0.479 ± 0.004	1.184 ± 0.003	27.98 ± 0.04	0.029 ± 0.000	0.199 ± 0.002	0.925 ± 0.001	0.556 ± 0.002
	up-DE	2	0.511 ± 0.001	14.07 ± 0.02	0.019 ± 0.000	**0.191** ± 0.001	**0.945** ± 0.000	0.694 ± 0.005	1.133 ± 0.002	26.97 ± 0.04	**0.022** ± 0.000	**0.449** ± 0.005	**0.861** ± 0.001	0.820 ± 0.003
	down-DE	4	0.518 ± 0.002	14.29 ± 0.03	0.022 ± 0.000	0.092 ± 0.001	0.969 ± 0.000	0.487 ± 0.003	1.154 ± 0.004	27.47 ± 0.05	0.023 ± 0.000	0.199 ± 0.002	0.925 ± 0.001	0.567 ± 0.002
	up-DE	4	**0.486** ± 0.000	**13.52** ± 0.02	**0.016** ± 0.000	**0.190** ± 0.001	**0.946** ± 0.000	**0.722** ± 0.001	**1.073** ± 0.001	**25.74** ± 0.02	0.030 ± 0.000	**0.446** ± 0.001	**0.862** ± 0.000	**0.857** ± 0.000
B/16	down-DE	2	0.519 ± 0.002	14.09 ± 0.02	0.021 ± 0.001	0.048 ± 0.000	0.982 ± 0.000	0.351 ± 0.002	1.316 ± 0.008	30.02 ± 0.18	0.030 ± 0.000	0.132 ± 0.001	0.943 ± 0.000	0.448 ± 0.002
	up-DE	2	0.489 ± 0.001	13.40 ± 0.03	**0.015** ± 0.000	**0.169** ± 0.002	**0.951** ± 0.000	0.668 ± 0.004	1.231 ± 0.004	28.41 ± 0.09	**0.023** ± 0.000	**0.481** ± 0.006	**0.845** ± 0.001	0.838 ± 0.003
	down-DE	4	0.511 ± 0.002	13.95 ± 0.01	0.019 ± 0.001	0.048 ± 0.000	0.982 ± 0.000	0.354 ± 0.002	1.293 ± 0.008	29.67 ± 0.18	0.026 ± 0.000	0.132 ± 0.001	0.943 ± 0.000	0.453 ± 0.002
	up-DE	4	**0.468** ± 0.000	**12.89** ± 0.03	**0.016** ± 0.000	**0.168** ± 0.000	**0.951** ± 0.000	**0.690** ± 0.001	**1.166** ± 0.002	**27.08** ± 0.05	0.037 ± 0.000	**0.479** ± 0.001	**0.846** ± 0.000	**0.879** ± 0.001
B/32	down-DE	2	0.620 ± 0.001	16.44 ± 0.04	0.023 ± 0.000	0.073 ± 0.001	0.973 ± 0.000	0.414 ± 0.002	1.510 ± 0.005	33.79 ± 0.08	0.032 ± 0.000	0.175 ± 0.001	0.925 ± 0.000	0.498 ± 0.002
	up-DE	2	0.588 ± 0.001	15.74 ± 0.05	**0.017** ± 0.001	**0.214** ± 0.001	**0.937** ± 0.000	0.709 ± 0.001	1.430 ± 0.003	32.37 ± 0.05	**0.022** ± 0.000	**0.537** ± 0.002	**0.824** ± 0.001	0.844 ± 0.002
	down-DE	4	0.607 ± 0.000	16.17 ± 0.02	**0.021** ± 0.001	0.073 ± 0.000	0.973 ± 0.000	0.418 ± 0.005	1.483 ± 0.008	33.36 ± 0.13	0.027 ± 0.000	0.174 ± 0.001	0.926 ± 0.001	0.504 ± 0.002
	up-DE	4	**0.561** ± 0.001	**15.10** ± 0.03	0.020 ± 0.000	**0.214** ± 0.001	**0.937** ± 0.000	**0.739** ± 0.001	**1.357** ± 0.002	**30.92** ± 0.03	0.036 ± 0.000	**0.537** ± 0.001	**0.824** ± 0.000	**0.884** ± 0.001
S/32	down-DE	2	0.807 ± 0.003	20.90 ± 0.10	**0.018** ± 0.001	0.102 ± 0.001	0.962 ± 0.000	0.458 ± 0.003	2.106 ± 0.010	44.52 ± 0.18	0.038 ± 0.001	0.223 ± 0.003	0.900 ± 0.001	0.521 ± 0.002
	up-DE	2	0.763 ± 0.001	19.85 ± 0.04	**0.016** ± 0.000	**0.305** ± 0.002	**0.911** ± 0.000	0.773 ± 0.002	2.004 ± 0.004	42.92 ± 0.08	**0.025** ± 0.000	**0.683** ± 0.003	**0.767** ± 0.001	0.856 ± 0.002
	down-DE	4	0.795 ± 0.003	20.66 ± 0.13	**0.015** ± 0.001	0.102 ± 0.002	0.962 ± 0.001	0.462 ± 0.004	2.076 ± 0.012	44.16 ± 0.21	0.031 ± 0.000	0.222 ± 0.003	0.900 ± 0.001	0.526 ± 0.003
	up-DE	4	**0.728** ± 0.001	**19.06** ± 0.04	0.025 ± 0.000	**0.304** ± 0.001	**0.911** ± 0.000	**0.808** ± 0.002	**1.914** ± 0.003	**41.38** ± 0.05	0.034 ± 0.000	**0.682** ± 0.003	**0.767** ± 0.001	**0.891** ± 0.002

C.9 FLOPs Numbers

Table C.13 provides the downstream training FLOPs for various E^3, V-MoE, and ViT configurations. These numbers correspond to the x-values of the points in the figures presented in Sections C.8 and 5.5. Table C.14 provides the percentage difference in FLOPs between the E^3, V-MoE and down-DE models most commonly used in this work. Note that the percentage differences for H/14 do not follow the trend of the other sizes, e.g. that the percentage difference between E^3 and V-MoE gets smaller for larger sizes, due to the fact that for H/15 we use a last-5 configuration rather than the last-2 configuration used for the other ViT families. Table C.15 provides the downstream training FLOPs for the various ablation study models presented in Section 5.4.2.[1]

Table C.13: Downstream training GFLOPs for the various E^3, V-MoE, and ViT baselines used in this work.

	K	M	S/32	B/32	B/16	L/32	L/16	H/14
E^3	1	2	36.69	105.89	452.62	320.92	1356.46	4183.28
	1	4	58.03	152.98	—	403.77	—	—
	2	2	47.98	131.06	552.65	365.42	1533.74	—
	2	4	80.66	203.31	—	492.77	—	—
	4	2	70.61	181.40	—	454.43	—	—
ViT	-	1	24.01	77.97	334.48	271.83	1151.71	3033.60
	-	2	48.02	155.93	668.95	543.66	2303.43	6067.21
	-	4	96.03	311.87	1337.90	1087.33	4606.85	12134.41
V-MoE	1	1	26.02	82.35	351.91	279.55	1182.08	3179.90
	1	2	52.04	164.70	703.81	559.10	2364.16	6359.81
	1	4	104.07	329.41	1407.63	1118.21	4728.32	12719.61
	2	1	31.66	94.93	401.98	301.75	1270.78	3617.94
	4	1	42.95	120.11	501.99	346.25	1448.06	—
	8	1	65.59	170.44	—	435.26	—	—

[1]Note that the E^3 and V-MoE results here are different to those in Table C.13 due to a difference in implementation. We used a simplified, but less computationally efficient, expert-routing implementation for all of the ablation studies. As a result, the V-MoE and E^3 FLOPs in Table C.13 are lower and cannot be fairly compared with the ablation models presented here. We thus re-benchmarked E^3 and V-MoE to obtain comparable results.

Table C.14: Percentage difference in downstream training FLOPs for E^3 with $(K, M) = (1, 2)$ compared with V-MoE with $K = 1$ and an ensemble of two such V-MoE members.

	S/32	B/32	B/16	L/32	L/16	H/14
E^3 vs. V-MoE	41.01	28.58	28.62	14.80	14.75	31.55
E^3 vs. down-DE	-29.49	-35.71	-35.69	-42.60	-42.62	-34.22

Table C.15: Downstream training GFLOPs comparison for the ablation study models in Section 5.4.2.

	K	M	GFLOPs
V-MoE	2	—	96.895
V-MoE	4	—	123.644
V-MoE	8	—	178.133
E^3	1	2	109.781
E^3	2	2	138.457
E^3	4	2	196.870
Tiling	2	2	138.460
Partitioning	2	—	97.885
Overlap = 2	2	2	138.457
Overlap = 4	2	2	138.458
Overlap = 8	2	2	138.459
Overlap = 6	2	2	138.460
Multi-pred	2	—	96.330
Multi-pred	4	—	122.526
Multi-pred	8	—	175.889

C.10 Additional Algorithm Overview Diagrams

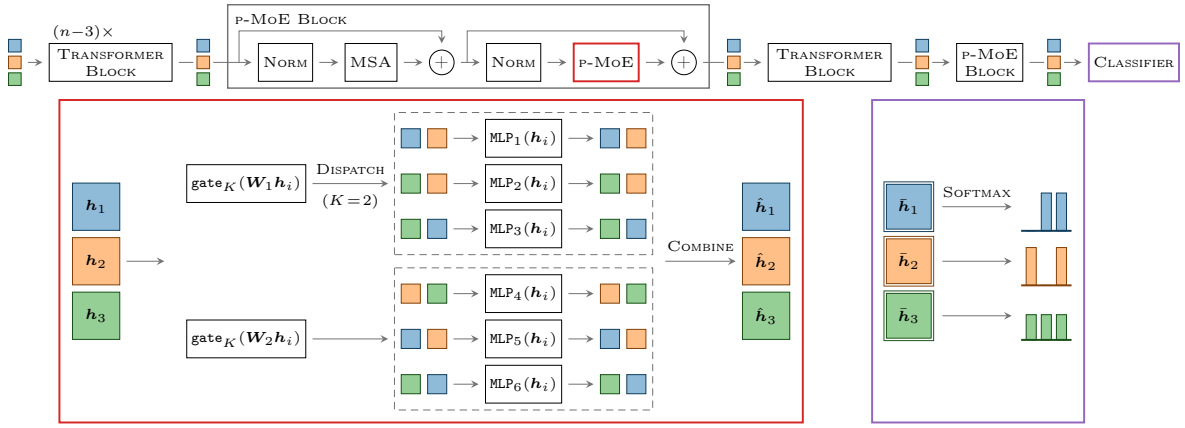


Figure C.13: End-to-end overview of the *Partitioning* method, from Section 5.4.2, with $E=6$ experts, partitioned into $M=2$ groups, with sparsity of $K=2$, and a "last-2" configuration. **Top**: *Partitioning* contains a sequence of transformer blocks, followed by alternating transformer and p(artitioned)-MoE blocks. As in ViT, images are split into patches whose embeddings are processed by each block. Here, we show 1 embedding for each of three images (■, ■, ■). **Bottom left**: In a MoE block, we replace the transformer block's MLP with parallel partitioned expert MLPs. The effect of the routing weights is not depicted. **Bottom right**: The classifier makes predictions from the final representations (▣). Notice that without a tiling mechanism, there is only a single prediction per input.

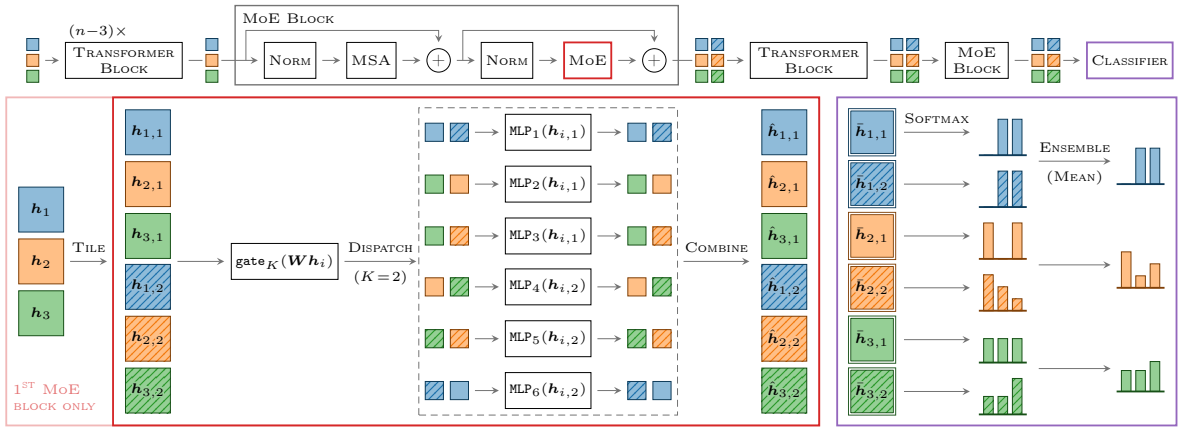


Figure C.14: End-to-end overview of the *Tiling* method, from Section 5.4.2, with $E = 6$ experts, a sparsity of $K = 2$, and a "last-2" configuration. **Top**: *Tiling* contains a sequence of transformer blocks, followed by alternating transformer and MoE blocks. As in ViT, images are split into patches whose embeddings are processed by each block. Here, we show 1 embedding for each of three images (■, ■, ■). **Bottom left**: In a MoE block, we replace the transformer block's MLP with parallel partitioned expert MLPs. The effect of the routing weights is not depicted. Embeddings are tiled (▨) in the first p-MoE block only. **Bottom right**: The classifier averages predictions from the final tiled representations (▣). Notice that without partitioning, some patches and their corresponding tiled versions can be routed to the same experts, resulting in a reduced diversity in predictions.

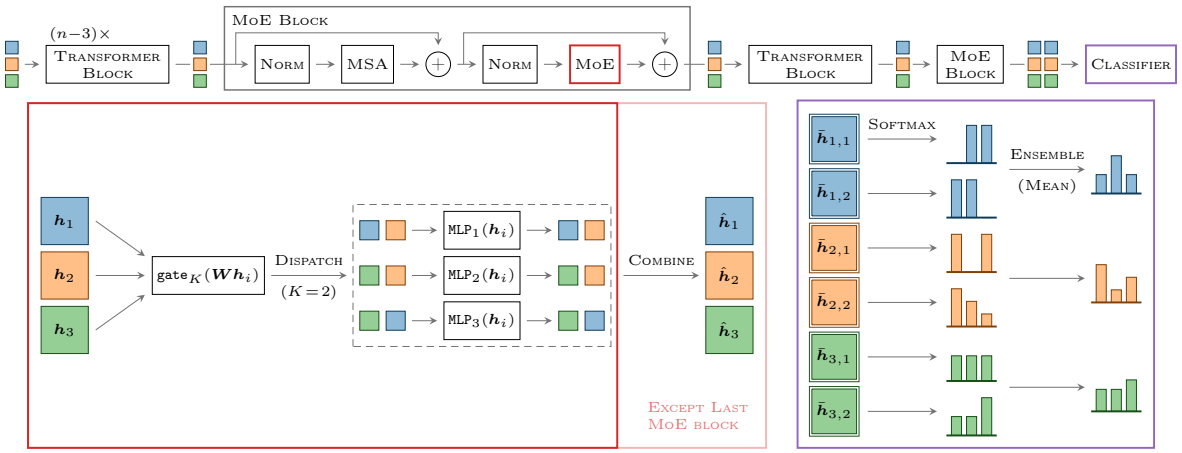


Figure C.15: End-to-end overview of the simple *Multi-pred* MoE, from Section 5.4.2, with $E = 3$ experts, sparsity of $K = 2$, and a "last-2" configuration. **Top**: The multi-pred MoE contains a sequence of transformer blocks, followed by alternating transformer and MoE blocks. As in ViT, images are split into patches whose embeddings are processed by each block. Here, we show 1 embedding for each of three images (■, ■, ■). **Bottom left**: In all but the last MoE block, we recombine the predictions as usual. **Bottom right**: The classifier averages predictions from the final representations (▣).

Appendix D

Generative Model of Symmetries Details

D.1 Connections to MLL Optimization

As we will now show, Algorithm 1 has connections to Marginal Log Likelihood (MLL) maximisation via Variational Autoencoder (VAE)-like amortised inference. Given the graphical model in Figure 6.2, we can derive an Evidence Lower BOund (ELBO) for jointly learning the generative and inference parameters with gradients:

$$\log p(\mathbf{x}) = \log \iint p(\mathbf{x}, \boldsymbol{\eta}, \hat{\mathbf{x}}) d\boldsymbol{\eta}\, d\hat{\mathbf{x}} \tag{D.1}$$

$$= \log \iint p(\mathbf{x} \mid \boldsymbol{\eta}, \hat{\mathbf{x}}) p_{\boldsymbol{\psi}}(\boldsymbol{\eta} \mid \hat{\mathbf{x}}) p_{\boldsymbol{\theta}}(\hat{\mathbf{x}}) d\boldsymbol{\eta}\, d\hat{\mathbf{x}}$$

$$= \log \iint p(\mathbf{x} \mid \boldsymbol{\eta}, \hat{\mathbf{x}}) p_{\boldsymbol{\psi}}(\boldsymbol{\eta} \mid \hat{\mathbf{x}}) p_{\boldsymbol{\theta}}(\hat{\mathbf{x}}) \frac{q_{\boldsymbol{\omega}}(\boldsymbol{\eta}, \hat{\mathbf{x}} \mid \mathbf{x})}{q_{\boldsymbol{\omega}}(\boldsymbol{\eta}, \hat{\mathbf{x}} \mid \mathbf{x})} d\boldsymbol{\eta}\, d\hat{\mathbf{x}} \tag{D.2}$$

$$= \log \mathop{\mathbb{E}}_{q_{\boldsymbol{\omega}}(\boldsymbol{\eta}, \hat{\mathbf{x}} \mid \mathbf{x})} \left[\frac{p(\mathbf{x} \mid \hat{\mathbf{x}}, \boldsymbol{\eta})\, p_{\boldsymbol{\psi}}(\boldsymbol{\eta} \mid \hat{\mathbf{x}})\, p_{\boldsymbol{\theta}}(\hat{\mathbf{x}})}{q_{\boldsymbol{\omega}}(\boldsymbol{\eta}, \hat{\mathbf{x}} \mid \mathbf{x})} \right] \tag{D.3}$$

$$\geq \underbrace{\mathop{\mathbb{E}}_{q_{\boldsymbol{\omega}}(\boldsymbol{\eta}, \hat{\mathbf{x}} \mid \mathbf{x})} [\log p(\mathbf{x} \mid \boldsymbol{\eta}, \hat{\mathbf{x}})]}_{\text{likelihood}} - \underbrace{D_{\mathrm{KL}} \left[q_{\boldsymbol{\omega}}(\boldsymbol{\eta}, \hat{\mathbf{x}} \mid \mathbf{x}) \,||\, p_{\boldsymbol{\psi}}(\boldsymbol{\eta} \mid \hat{\mathbf{x}}) p_{\boldsymbol{\theta}}(\hat{\mathbf{x}}) \right]}_{\text{KL-divergence}} \tag{D.4}$$

$$\equiv -\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\psi}, \boldsymbol{\omega}), \tag{D.5}$$

where $p_{\boldsymbol{\theta}}(\hat{\mathbf{x}})$ is some generative model—e.g., a VAE—for prototypes, with parameters $\boldsymbol{\theta}$, and $q_{\boldsymbol{\omega}}(\boldsymbol{\eta}, \hat{\mathbf{x}} \mid \mathbf{x}) = q_{\boldsymbol{\omega}}(\boldsymbol{\eta} \mid \mathbf{x}) p(\hat{\mathbf{x}} \mid \mathbf{x}, \boldsymbol{\eta})$. Now, we can show that the gradient of the *likelihood* term in the ELBO is approximated by the gradient of our Self-Supervised

Learning (SSL) loss on Line 1 of Algorithm 1:

$$\nabla_{\omega} \mathop{\mathbb{E}}_{q_\omega(\eta \mid \mathbf{x}) p(\hat{\mathbf{x}} \mid \mathbf{x}, \eta)} \left[\log p\left(\mathbf{x} \mid \hat{\mathbf{x}}, \eta\right)\right] \tag{D.6}$$

▷ $p\left(\mathbf{x} \mid \hat{\mathbf{x}}, \eta\right) = \delta\left(\mathbf{x} - \mathcal{T}_\eta(\hat{\mathbf{x}})\right) = \lim_{\sigma^2 \to 0} \mathcal{N}(\mathbf{x} \mid \mathcal{T}_\eta(\hat{\mathbf{x}}), \sigma^2)$:

$$\approx \nabla_{\omega} \mathop{\mathbb{E}}_{q_\omega(\eta \mid \mathbf{x}) p(\hat{\mathbf{x}} \mid \mathbf{x}, \eta)} \left[\log \mathcal{N}\left(\mathbf{x} \,\middle|\, \mathcal{T}_\eta(\hat{\mathbf{x}}), \sigma^2\right)\right] \tag{D.7}$$

▷ take 1 sample, $\boldsymbol{\eta} \sim q_\omega(\eta \mid \boldsymbol{x})$:

$$\approx \nabla_{\omega} \log \mathcal{N}\left(\boldsymbol{x} \,\middle|\, \mathcal{T}_{\boldsymbol{\eta}}(\hat{\boldsymbol{x}}), \sigma^2\right), \tag{D.8}$$

▷ definition of Gaussian PDF:

$$= \nabla_{\omega} \text{-} 0.5 \left\|\boldsymbol{x} - \mathcal{T}_{\boldsymbol{\eta}}\left(\hat{\boldsymbol{x}}\right)\right\|_2^2 / \sigma^2 - \log\left(\sqrt{2\pi}\sigma\right) \tag{D.9}$$

▷ drop constant term:

$$= \nabla_{\omega} \text{ -} 0.5 \, \texttt{mse}\left(\boldsymbol{x}, \mathcal{T}_{\boldsymbol{\eta}}\left(\hat{\boldsymbol{x}}\right)\right) / \sigma^2. \tag{D.10}$$

The negative sign is due to the fact that the ELBO is maximised, whereas our SSL loss is minimised. The gradient of the *KL-divergence* term w.r.t. $\boldsymbol{\psi}$ is approximated by the gradient of our MLE loss on Line 8 of Algorithm 1:

$$\nabla_{\psi} D_{\mathrm{KL}}\left[q_\omega(\eta, \hat{\mathbf{x}} \mid \mathbf{x}) \,||\, p_\psi(\eta \mid \hat{\mathbf{x}}) p_\theta(\hat{\mathbf{x}})\right] \tag{D.11}$$

▷ definition of D_{KL}:

$$= \nabla_{\psi} \mathop{\mathbb{E}}_{q_\omega(\eta \mid \mathbf{x}) p(\hat{\mathbf{x}} \mid \mathbf{x}, \eta)} \left[\log \frac{q_\omega(\eta \mid \mathbf{x}) p\left(\hat{\mathbf{x}} \mid \mathbf{x}, \eta\right)}{p_\psi(\eta \mid \hat{\mathbf{x}}) p_\theta(\hat{\mathbf{x}})}\right] \tag{D.12}$$

▷ drop constant terms and use $\hat{\mathbf{x}} = \mathcal{T}_\eta^{-1}(\mathbf{x})$:

$$= \nabla_{\psi} \mathop{\mathbb{E}}_{q_\omega(\eta \mid \mathbf{x})} \left[\text{-} \log p_\psi\left(\eta \,\middle|\, \mathcal{T}_\eta^{-1}(\mathbf{x})\right)\right] \tag{D.13}$$

▷ take 1 sample, $\boldsymbol{\eta}_x \sim q_\omega(\eta \mid \mathbf{x})$:

$$\approx \nabla_{\psi} \text{-} \log p_\psi\left(\boldsymbol{\eta}_{\boldsymbol{x}} \,\middle|\, \mathcal{T}_{\boldsymbol{\eta}_{\boldsymbol{x}}}^{-1}(\boldsymbol{x})\right). \tag{D.14}$$

Note that the sampling approximations in both (D.8) and (D.14) also apply to VAE-like amortised inference algorithms.

While ELBO training and our algorithm share some similarities, some key differences exist. For instance, we do not learn the generative and inference models jointly. This disjoint training is equivalent to ignoring the gradient $\nabla_\omega D_{\mathrm{KL}}\left[q_\omega(\eta, \hat{\mathbf{x}} \mid \mathbf{x}) \,||\, p_\psi(\eta \mid \hat{\mathbf{x}}) p_\theta(\hat{\mathbf{x}})\right]$ when training $q_\omega(\eta \mid \mathbf{x})$. This KL-divergence has two components: entropy $\text{-}\mathbb{H}\left[q_\omega\right]$ and

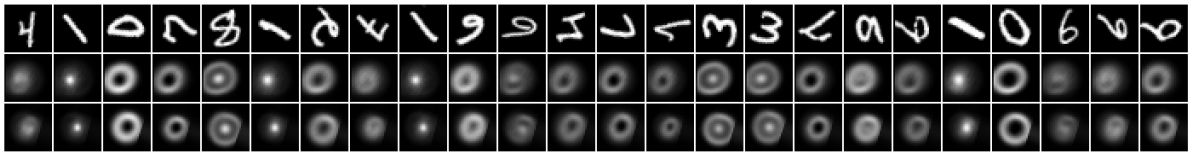


Figure D.1: Failure of an invariant VAE encoder. **Top:** MNIST digits sampled from the test set. **Mid:** Prototypes produced by VAE who's encoder is made invariant using (D.15), where $\boldsymbol{\eta} \sim \mathcal{U}(-\boldsymbol{\eta}_{\max}, \boldsymbol{\eta}_{\max})$ and $\boldsymbol{\eta}_{\max} = (0.25, 0.25, \pi, 0.25, 0.25)$. **Bot:** Reconstructed digits. The model becomes stuck in a local optima where the prototypes and 'reconstructions' are all circles and rings of various sizes, depending on the input image. The averaged latent code is free of (e.g.,) rotation information but has also lost almost all information that identifies each digit.

cross entropy $\mathbb{H}[q_\omega, p_\psi p_\theta]$. Assuming that $p_\psi(\boldsymbol{\eta} \,|\, \hat{\mathbf{x}})$ is sufficiently flexible, the cross entropy term should not have a significant impact on $q_\omega(\boldsymbol{\eta} \,|\, \mathbf{x})$ since p_ψ is trained to match q_ω. On the other hand, $q_\omega(\boldsymbol{\eta} \,|\, \mathbf{x})$ should be close to a delta since there should be a single prototype for each $\mathbf{x}$. Thus, encouraging high variance with an entropy term might actually be harmful. Another difference is that we do not need to learn $p_\theta(\hat{\mathbf{x}})$, which has the benefit that we can learn the symmetries in a dataset without having to learn to generate the data itself, greatly simplifying training for complicated datasets. Furthermore, actually evaluating the gradient of the likelihood term in (D.5) is challenging due to the fact that $p(\mathbf{x} \,|\, \hat{\mathbf{x}}, \boldsymbol{\eta})$ is a delta.

Given all of these differences, it might be natural to question the utility of the comparison between our algorithm and maximisation of (D.5). Perhaps the most useful connection to draw is that of (D.11) to (D.14), which motivates our MLE learning objective for $p_\omega(\boldsymbol{\eta} \,|\, \hat{\mathbf{x}})$ as being closely related to the process of learning a prior in an ELBO.

In an early version of this work (Allingham et al., 2022a), we trained a variant of our Symmetry-aware Generative Model (SGM) using an ELBO similar to the one above, with the main difference being that $\hat{\mathbf{x}}$ was modelled using a VAE and invariance was incorporated into the VAE encoder. We constructed an invariant encoder $q_\phi(\mathbf{z} \,|\, \mathbf{x})$ from a non-invariant encoder $\hat{q}_\phi(\mathbf{z} \,|\, \mathbf{x})$:

$$q_\phi(\mathbf{z} \,|\, \mathbf{x}) \equiv \mathbb{E}_\eta \left[\hat{q}_\phi(\mathbf{z} \,|\, \mathbf{x})\right], \tag{D.15}$$

following Benton et al. (2020); van der Ouderaa and van der Wilk (2022); Immer et al. (2022). We found that this approach worked well under a single transformation (e.g., rotation) but that it quickly broke down as the space of transformations was expanded (e.g., to all affine transformations; see Figure D.1). We hypothesise that the averaging of many latent codes makes it difficult to learn an invariant representation $\mathbf{z}$ without

throwing away *all* of the information in $\mathbf{x}$. This further motivates our SSL algorithm for learning invariant prototypes. A similar observation was also made by Dubois et al. (2021), who found that an SSL-based objective was superior to an ELBO-based method for learning invariant representations in the context of compression.

D.2 Further Practical Considerations

In this section, we elaborate on Section 6.3.1 and provide additional considerations.

Learning $q_{\boldsymbol{\omega}}(\boldsymbol{\eta}|\mathbf{x})$ instead of $f_{\boldsymbol{\omega}}$. We found that learning $f_{\boldsymbol{\omega}}$ probabilistically—i.e., allowing for some uncertainty in the transformation during the training process by parameterising a density function over $\mathcal{H}$ with $q_{\boldsymbol{\omega}}(\boldsymbol{\eta}|\mathbf{x})$ and sampling $\boldsymbol{\eta}$—provides small improvements in performance. The distribution $q_{\boldsymbol{\omega}}(\boldsymbol{\eta}|\mathbf{x})$ quickly collapses to a delta, thus, we hypothesise that the added noise from sampling acts as a regulariser that is helpful at the start of training.

Inference network blurring schedule. Occasionally, depending on the dataset, random seed, kind of transformations being applied, and other hyperparameters, training the inference network fails, and the prototype transformations would be 100% lossy—i.e., they would result in completely empty images—regardless of the strength of the invertibility loss. We found that we could prevent this from happening by adding a small amount of Gaussian blur to each example. Furthermore, we found that we only needed to add this blur for a small fraction of the initial training steps to prevent the model from falling into this degenerate local optima.

Averaging multiple samples for the SSL loss. Just as we found averaging the MLE loss over multiple samples to improve performance, so too is averaging the SSL loss.

We compare rotation inference nets—with hidden layers of dimensions $[2048, 1024, 512, 256, 128]$ trained for 2k steps using the AdamW optimizer with a cosine decayed with warmup learning rate schedule that starts at 1×10^{-4}, increases to 3×10^{-4} in 500 steps, and then decreases to 1×10^{-7}, with a batch size of 256—trained on fully rotated MNIST digits using the SSL objective averaged over 1, 3, 5, 10, and 30 samples:

Samples	**x**-mse
1	0.0981
3	0.0901
5	**0.0840**
10	0.0853
30	0.0870

As the number of samples increases, **x**-mse decreases until saturating around 5 samples. Note that this relationship is not likely to be *monotonically* decreasing because there is random noise in each training run (i.e., due to random NN initialisation, etc.). That said, we expect it will decrease on average as the number of samples increases. We find 5 samples to be a good trade-off between improved performance and increased compute.

Symmetric SSL loss. In our SSL loss, based on Figure 6.4, we are essentially comparing the prototypes given $\boldsymbol{x}$ and $\boldsymbol{x}_{\text{rnd}}$ (a randomly transformed version of $\boldsymbol{x}$). An alternative is to compare the prototypes given $\boldsymbol{x}_{\text{rnd1}}$ and $\boldsymbol{x}_{\text{rnd2}}$, two randomly transformed versions of $\boldsymbol{x}$:

$$\left\| \mathcal{T}^{-1}_{f_{\boldsymbol{\omega}}(\boldsymbol{x}_{\text{rnd1}})}(\boldsymbol{x}_{\text{rnd1}}) - \mathcal{T}^{-1}_{f_{\boldsymbol{\omega}}(\boldsymbol{x}_{\text{rnd2}})}(\boldsymbol{x}_{\text{rnd2}}) \right\|_2^2, \;\; \boldsymbol{x}_{\text{rnd1}} = \mathcal{T}_{\boldsymbol{\eta}_{\text{rnd1}}}(\boldsymbol{x}), \tag{D.16}$$

where $\boldsymbol{x}_{\text{rnd2}} = \mathcal{T}_{\boldsymbol{\eta}_{\text{rnd2}}}(\boldsymbol{x}),\;\; \boldsymbol{\eta}_{\text{rnd1}}, \boldsymbol{\eta}_{\text{rnd2}} \sim p(\boldsymbol{\eta}_{\text{rnd}})$. As before, we modify this loss to allow us to compose transformations to get

$$\left\| \mathcal{T}_{f_{\boldsymbol{\omega}}(\boldsymbol{x}_{\text{rnd2}})} \circ \mathcal{T}^{-1}_{f_{\boldsymbol{\omega}}(\boldsymbol{x}_{\text{rnd}})}(\boldsymbol{x}_{\text{rnd}}) - \boldsymbol{x}_{\text{rnd2}} \right\|_2^2. \tag{D.17}$$

The motivation for using this 'symmetric' SSL loss is that it provides the inference network with additional data augmentation—the inference network is now unlikely to ever see the $\boldsymbol{x}$ twice. We find that while this works well for MNIST, it *does not* work well for dSprites. This is because the transformations in dSprites in dSprites are more lossy than those for MNIST. E.g., it is easier to shift a small sprite out of the frame of an image compared to a large digit. Thus, the symmetric loss results in a much higher variance when used with dSprites, which negatively impacts training.

Composing affine transformations of images. Care must be taken when composing affine transformations of images when implemented via a coordinate transformation (e.g., `affine_grid` & `affine_sample` in PyTorch, or `scipy.map_coords` in Jax). To compose two affine transformations parameterised by $\boldsymbol{\eta}_1$ and $\boldsymbol{\eta}_2$, the affine matrices $T(\boldsymbol{\eta}_1), T(\boldsymbol{\eta}_2)$

need to be *right*-multiplied with one another; in other words $\mathcal{T}_{\boldsymbol{\eta}_2} \circ \mathcal{T}_{\boldsymbol{\eta}_1} = \mathcal{T}'_{T(\boldsymbol{\eta}_1)T(\boldsymbol{\eta}_2)}$. This is because, in these implementations of affine transformation of images, the affine transformation is applied to the pixel grid (i.e., the reference frame), rather than to the image itself. In effect, the resulting transformation as applied to the objects in the image is the opposite; if the reference frame moves to the right, the objects in the image move to the left, etc. More concretely, when the reference frame is affine-transformed by $\mathcal{T}$, the image itself is affine-transformed by $\mathcal{T}^{-1}$.

Overfitting of the generative network. While we did not observe any overfitting of the inference network (likely due to the built-in 'data augmentation' of our SSL loss, and the general difficulty of learning a function with equivariance to arbitrary transformations), we did find that the generative network is prone to overfitting. We address this by using a validation set to optimise several relevant hyperparameters (e.g., dropout rates, number of flow layers, number of training epochs, etc.); see Section D.3.

Learning $p_{\boldsymbol{\psi}}(\boldsymbol{\eta} \mid \hat{\mathbf{x}})$ with imperfect inference, continued. To encourage $p_{\boldsymbol{\psi}}(\boldsymbol{\eta} \mid \hat{\mathbf{x}})$ produce the same distribution for the inconsistent prototypes produced by $q_{\boldsymbol{\omega}}(\boldsymbol{\eta} \mid \mathbf{x})$, we add a *consistency* loss to Line 8 of Algorithm 1 the MLE objective:

$$L_{\text{consistency}}(\boldsymbol{\psi}) = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} |\log p_i - \log p_j|, \tag{D.18}$$

where $p_i = p_{\boldsymbol{\psi}}(\boldsymbol{\eta}_{\boldsymbol{x}} \mid \hat{\boldsymbol{x}}'_i)$ and $\hat{\boldsymbol{x}}'_i$ is due to the i^{th} $\boldsymbol{\eta}_{\text{rnd}}$ sample.

D.3 Experimental Setup

We use `jax` with `flax` for NNs, `distrax` for probability distributions, and `optax` for optimisers. We use `ciclo` with `clu` to manage our training loops, `ml_collections` to specify our configurations, and `wandb` to track our experiments. The code is available at https://github.com/cambridge-mlg/sgm.

Unless otherwise specified, we use the following NN architectures and other hyperparameters for all of our experiments. We use the AdamW optimiser with weight decay of 1×10^{-4}, global norm gradient clipping, and a linear warm-up followed by a cosine decay as a learning rate schedule. The exact learning rates and schedules are discussed for each model below. We use a batch size of 512.

All of our Multi-Layer Perceptrons (MLPs) use `gelu` activations, and `LayerNorm`. In some cases, we use `Dropout`. The structure of each layer is `Dense` $\rightarrow$ `gelu` $\rightarrow$

`LayerNorm` → `Dropout`. Whenever we learn or predict a scale parameter σ, it is constrained to be positive using a `softplus` operation.

Inference network. We use a MLP with hidden layers of dimension $[2048, 1024, 512, 256]$. The network outputs a mean $\boldsymbol{\eta}$ prediction for each example, and the uncertainty—as mentioned in Section D.2—is implemented as a homoscedastic scale parameter. We train for 60k steps. For each example, we average the loss over 5 random augmentations. In some settings—also mentioned in Section D.2—we add a small amount of blur to the images with a Gaussian filter of size 5, for the first 1% of training steps. The σ value for the filter was linearly decayed from its maximum to 0. The initial maximum value is specified below.

Generative network. Our generative model is a Neural Spline Flow (NSF) (Durkan et al., 2019) with 6 bins in the range $[-3, 3]$. We use a MLP with hidden layers of dimension $[1024, 512, 512]$ as a shared feature extractor. The base normal distribution's mean and scale parameters are predicted by another MLP, with hidden layers of dimension $[256, 256]$, whose input is the shared feature representation. The parameters of the spline at each layer of the Normalising Flow (NF) are predicted by MLPs with a single hidden layer of dimension 256, with a dropout rate of 0.1, whose input is a concatenation of the shared feature representation, and the (masked) outputs of the previous layer. For each example, we average the loss over 5 random augmentations.

D.3.1 MNIST under affine transformations

We split the MNIST training set by removing the last 10k examples and using them exclusively for validation and hyperparameter sweeps. We randomly augment the inputs by sampling transformation parameters from $\mathcal{U}(-\boldsymbol{\eta}_{\max}, \boldsymbol{\eta}_{\max})$, where $\boldsymbol{\eta}_{\max} = (0.25, 0.25, \pi, 0.25, 0.25)$ is the maximum (x-shift, y-shift, rotation, x-scale, y-scale) applied to the images. All affine transformations are applied with bicubic interpolation.

Inference network. The invertibility loss $\mathcal{L}_{\text{invertibility}}$ (6.7) is multiplied by a factor of 0.1. For the VAE data-efficiency results in Figure 6.11, we performed the following hyperparameter grid search for each random seed and amount of training data:

- blur $\sigma_{\text{init}} \in [0, 3]$,
- gradient clipping norm $\in [3, 10]$,

- learning rate $\in [1\times10^{-3}, 3\times10^{-4}, 1\times10^{-4}]$,
- initial learning rate multiplier $\in [3\times10^{-2}, 1\times10^{-2}]$,
- final learning rate multiplier $\in [1\times10^{-3}, 3\times10^{-4},]$, and
- warm-up steps % $\in [0.05, 0.1, 0.2]$.

All of the other MNIST affine transformation results use a blur σ_{init} of 0, a gradient clipping norm of 10, a learning rate of 3×10^{-4}, an initial learning rate multiplier of 1×10^{-2}, a final learning rate multiplier of 1×10^{-3}, and a warm-up steps % of 0.2, which are the best hyperparameters for 50k training examples with an arbitrarily chosen random seed. We use the 'symmetric' SSL loss discussed in Section D.2.

Generative network. We use an initial learning rate multiplier of 0.1, a gradient clipping norm of 2, and a warm-up steps % of 0.2. For the VAE data-efficiency results in Figure 6.11, we performed the following hyperparameter grid search for each random seed and amount of training data:

- learning rate $\in [3\times10^{-3}, 3\times10^{-4}]$,
- final learning rate multiplier $\in [0.3, 0.03]$,
- number of training steps $\in [7.5\text{k}, 15\text{k}, 30\text{k}, 60\text{k}]$,
- number of flow layers $\in [4, 5, 6]$,
- shared feature extractor dropout rate $\in [0.05, 0.1, 0.2]$, and
- consistency loss multiplier $\in [0, 1]$ (whether or not to use (D.18)).

When sweeping over the generative network hyperparameters, we require a trained inference network. We use the inference network hyperparameters for the same (random seed, number of training examples) pair. All of the other MNIST affine transformation results use a learning rate of 3×10^{-3}, a final learning rate multiplier of 0.03, 60k training steps, 6 flow layers, and a dropout rate of 0.2 in the shared feature extractor, which are the best hyperparameters for 50k training examples.

D.3.2 MNIST under colour transformations

For colour transformation on the MNIST dataset, we follow the same setup as above, with the following exceptions. We do not use an invertibility loss when training the inference network. Instead, for both the inference and generative networks, we constrain the outputs to be in $[-\boldsymbol{\eta}_{\max}, \boldsymbol{\eta}_{\max}]$, where $\boldsymbol{\eta}_{\max} = (0.5, 2.301, 0.51)$ using with `tanh` and `scale` bijectors.

Inference network. We use a blur σ_{init} of 3, a gradient clipping norm of 2, a learning rate of 3×10^{-4}, an initial learning rate multiplier of 1×10^{-2}, a final learning rate multiplier of 1×10^{-4}, and a warm-up steps % of 0.1, which were chosen using the same grid sweep as MNIST with affine transformations.

Generative network. We use a learning rate of 3×10^{-3}, with an initial learning rate multiplier of 1×10^{-1}, a final learning rate multiplier of 3×10^{-2}, 15k training steps, 6 flow layers, and a dropout rate of 0.2 in the shared feature extractor.

D.3.3 dSprites under affine transformations

For our dSprites experiments, we follow the same setup as for MNIST above, with the following exceptions. We do not use an invertibility loss when training the inference network. Instead, for both the inference and generative networks, we constrain their outputs to be in $[-\boldsymbol{\eta}_{\max}, \boldsymbol{\eta}_{\max}]$, where $\boldsymbol{\eta}_{\max} = (0.75, 0.75, \pi, 0.75, 0.75)$ using with `tanh` and `scale` bijectors. We *do not* use the 'symmetric' SSL loss discussed in Section D.2.

Inference network. We randomly augment the inputs by sampling transformation parameters from $\mathcal{U}(-\boldsymbol{\eta}_{\max}, -\boldsymbol{\eta}_{\max})$, where $\boldsymbol{\eta}_{\max}$ matches the constraints above. We use a blur σ_{init} of 3, a gradient clipping norm of 3, a learning rate of 1×10^{-3}, an initial learning rate multiplier of 3×10^{-2}, a final learning rate multiplier of 1×10^{-3}, and a warm-up steps % of 0.05, which were chosen using the same grid sweep as MNIST with affine transformations.

Generative network. We randomly augment the inputs by sampling transformation parameters from $\mathcal{U}(-\boldsymbol{\eta}_{\max} \times 0.75, -\boldsymbol{\eta}_{\max} \times 0.75)$, where $\boldsymbol{\eta}_{\max}$ matches the constraints above. We use a learning rate of 3×10^{-4}, a final learning rate multiplier of 0.3, 60k training steps, 6 flow layers, and a dropout rate of 0.05 in the shared feature extractor, which were chosen using the same grid sweep as MNIST with affine transformations.

Although we swept over the consistency loss multiplier, we accidentally always used a consistency loss multiplier of 1 in our experiments. This means that for some (random seed, amount of training data) pairs, the performance of our generative network is slightly lower than it should be, since the chosen hyperparameters may correspond to a consistency loss multiplier of 0. We include this detail for reproducibility, but note that it does not change our findings in any material way.

dSprites Setup

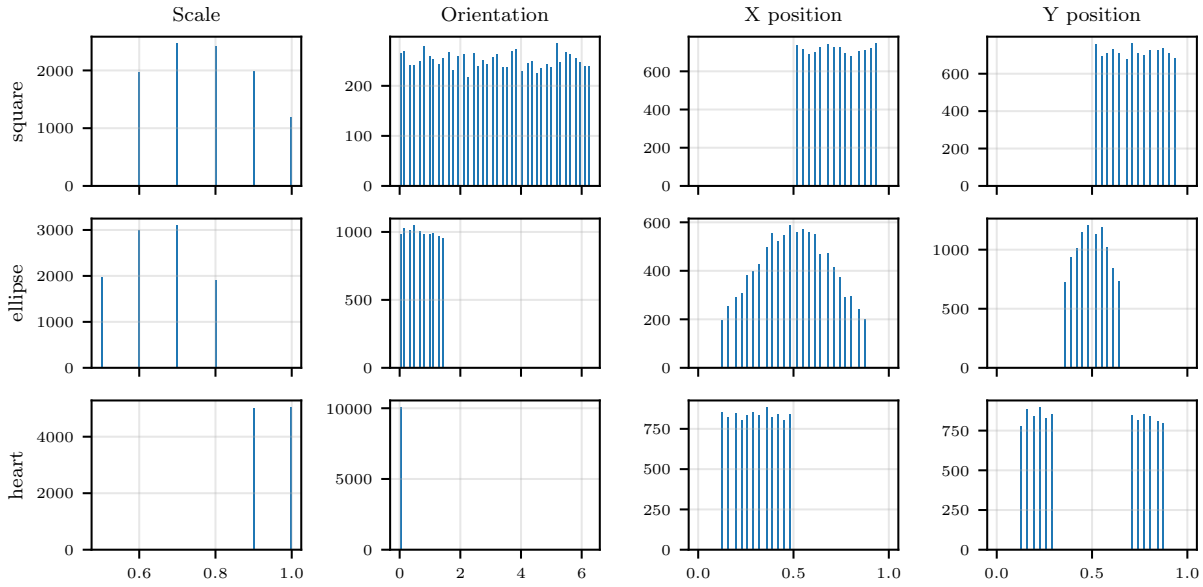


Figure D.2: Latent factor distributions for our modified dSprites data loader.

The original dSprites dataset contains sprites with the following factors of variation (Matthey et al., 2017).

- Colour: white
- Shape: square, ellipse, heart
- Scale: 6 values linearly spaced in $[0.5, 1]$
- Orientation: 40 values linearly spaced in $[0, 2\pi]$
- X position: 32 values linearly spaced in $[0, 1]$
- Y position: 32 values linearly spaced in $[0, 1]$

The dataset consists of sprites with the outer product of these factors, for a total of 737280 examples. We modified our data loader to resample the sprites proportional to the following distributions on the latent latent factors, conditioned on the shape.

- **square**
  - Scale: $\text{TruncNorm}\,(\mu = 0.75,\, \sigma^2 = 0.2,\, \text{min} = 0.55,\, \text{max} = 1.0)$
  - Orientation: $\mathcal{U}(0.0,\, 2\pi)$
  - X position: $\mathcal{U}(0.5,\, 0.95)$
  - Y position: $\mathcal{U}(0.5,\, 0.95)$
- **ellipse**
  - Scale: $\text{TruncNorm}\,(0.65,\, 0.15,\, 0.5,\, 0.85)$
  - Orientation: $\mathcal{U}(0.0,\, \pi/2)$
  - X position: $\text{TruncNorm}\,(0.5,\, 0.25,\, 0.1,\, 0.9)$
  - Y position: $\text{TruncNorm}\,(0.5,\, 0.15,\, 0.35,\, 0.65)$
- **heart**
  - Scale: $\mathcal{U}(0.9,\, 1.0)$
  - Orientation: $\delta\,(0.0)$
  - X position: $\mathcal{U}(0.1,\, 0.5)$
  - Y position: $0.5 \cdot \mathcal{U}(0.1,\, 0.3) + 0.5 \cdot \mathcal{U}(0.7,\, 0.9)$

An example of the resulting empirical distributions over the latent factors is shown in Figure D.2. The three shapes are sampled with equal proportions.

D.3.4 GalaxyMNIST under affine and colour transformations

For our GalaxyMNIST experiments, we follow the same setup as for MNIST under affine transformations above, with the following exceptions. We do not use an invertibility loss when training the inference network. Instead, for both the inference and generative networks, we constrain their outputs to be in $[-\boldsymbol{\eta}_{\text{max}},\, \boldsymbol{\eta}_{\text{max}}] + (0., 0., 0., 0., 0., 0.5, 0., 0.)$, where $\boldsymbol{\eta}_{\text{max}} = (0.75, 0.75, \pi, 0.75, 0.75, 0.5, 2.31, 0.51)$ using with `tanh` and `scale` bijectors. This dataset contains 10k examples. We use the last 2k as our test set, and the previous 1k as a validation set.

Inference network. We use an MLP with hidden layers of dimension $[1024, 1024, 512, 256]$. We train for 10k steps. We randomly augment the inputs by sampling transformation parameters from $\mathcal{U}(-\boldsymbol{\eta}_{\max} + (0., 0., 0., 0., 0., 0.5, 0., 0.), \boldsymbol{\eta}_{\max} + (0., 0., 0., 0., 0., 0.5, 0., 0.))$, where $\boldsymbol{\eta}_{\max}$ matches the constraints above. For the VAE data-efficiency results in Figure 6.12, we performed the same hyperparameter grid search as above for each random seed and amount of training data. All of the other GalaxyMNIST results use a blur σ_{init} of 0, a gradient clipping norm of 10, a learning rate of $3{\times}10^{-4}$, an initial learning rate multiplier of $1{\times}10^{-2}$, a final learning rate multiplier of $3{\times}10^{-4}$, and a warm-up steps % of 0.2, which are the best hyperparameters for 7k training examples with an arbitrarily chosen random seed. We use the 'symmetric' SLL loss discussed in Section D.2.

Generative network. We randomly augment the inputs by sampling transformation parameters from $\mathcal{U}(-\boldsymbol{\eta}_{\max} \times 0.75 + (0., 0., 0., 0., 0., 0.5, 0., 0.), \boldsymbol{\eta}_{\max} \times 0.75 + (0., 0., 0., 0., 0., 0.5, 0., 0.))$, where $\boldsymbol{\eta}_{\max}$ matches the constraints above. For the VAE data-efficiency results in Figure 6.12, we perform the same hyperparameter grid search as above for each random seed and amount of training data, with the following changes.[1] The sweep for number of training steps is $[3.75\text{k}, 7.5\text{k}, 15\text{k}]$. All of the other GalaxyMNIST results use a learning rate of $3{\times}10^{-4}$, a final learning rate multiplier of 0.03, 15k training steps, 4 flow layers, a dropout rate of 0.05 in the shared feature extractor, and a consistency loss multiplier of 1, which were chosen using the same grid sweep for an arbitrary random seed and 7k training examples.

D.3.5 PatchCamelyon under affine and colour transformations

We resized the images from 96×96 pixels to 64×64 using bilinear interpolation. The dataset has dedicated train, test, and validation splits, which we use without any modifications.

We follow the same setup as for GalaxyMNIST under affine and colour transformations above, with the exceptions listed below. We only used a single random seed.

Inference network. We train for 20k steps.

Generative network. The sweep for number of training steps is $[15\text{k}, 30\text{k}, 60\text{k}]$.[2]

[1] Our GalaxyMNIST results have the same issue as our dSprites results—the sweep included a consistency loss multiplier which was always set to a value of 1 in our experiments. This results in some small performance degradations.

[2] Our PatchCamelyon results have the same consistency multiplier issue as our dSprites and GalaxyMNIST results.

D.3.6 VAE, AugVAE, and InvVAE

Our VAEs use a latent code size of 20. The prior is a normal distribution with learnable mean and scale, initialised to 0s and 1s, respectively.

Our VAE encoders are LeNet-style CNNs with convolutional feature extractors followed by a MLP with a single hidden layer of size 256. The convolutional feature extractors use `gelu` activations and `LayerNorm`. The structure is `Conv` $\rightarrow$ `gelu` $\rightarrow$ `LayerNorm`. All `Conv` layers use 3×3 filters. The first two `Conv` have a stride of 2, while all others have a stride of 1. In between the convolutional layers and the MLP, there is a special dimensionality reduction `Conv` with only 3 filters followed by a `flatten`. For each dimension of the latent code, the encoder predicts a mean μ and a scale σ. The means and scales are initialised to 0s and 1s, respectively.

Our VAE decoders are inverted versions of our encoders. That is, we reverse the order of all of the `Dense` and `Conv` layers. The dimensionality reduction `Conv` layer and the `flatten` operation are replaced with the appropriate `Dense` layer and `reshape` operation. We replace all other `Conv` layers with `ConvTransposed` layers For each pixel of an image, the decoder predicts a mean μ. We learn a homoscedastic per-pixel scale σ. The scales are initialised to 1.

We use an initial learning rate multiplier of 3×10^{-2}, and a final learning rate multiplier of 1×10^{-4}. We run the following grid sweep for each (random seed, number of training examples, maximum added rotation angle) triplet:

- learning rate $\in [3\times10^{-3}, 6\times10^{-3}, 9\times10^{-3}]$,
- convolutional filters $\in [(64, 128), (64, 128, 256)]$,
- number of training steps $\in [5\text{k}, 10\text{k}, 20\text{k}]$, and
- warm-up steps % $\in [0.15, 0.2]$.

When running the sweep for AugVAE and InvVAE we use the inference and generative network hyperparameters for the same (random seed, number of training examples) pair.

PatchCamelyon

For our PatchCamelyon experiments, we use only a single random seed and a slightly modified hyperparameter sweep:

- learning rate $\in [3\times10^{-3}, 6\times10^{-3}$,
- convolutional filters $\in [(64, 128), (64, 128, 256), (128, 256, 512)]$,

- number of dense hidden layers $\in [1, 2]$,
- number of training steps $\in [20\text{k}, 30\text{k}, 40\text{k}]$, and
- warm-up steps % $\in [0.15]$.

D.3.7 Parametrisations of Symmetry transformations

We consider 5 affine transformations: shift in x, shift in y, rotation, scaling in x, and scaling in y. We represent these transformations using affine transformation matrices $\boldsymbol{A} = \exp\left(\sum_i \eta_i \boldsymbol{G}_i\right)$, where $\boldsymbol{G}_i$ are generator matrices for rotation, translation, and scaling; see Benton et al. (2020). The transformations are applied to an image by transforming the coordinates $(x,\ y)$ of each pixel, as in Jaderberg et al. (2015): $\begin{bmatrix} x' & y' & 1 \end{bmatrix}^\intercal = \boldsymbol{A} \cdot \begin{bmatrix} x & y & 1 \end{bmatrix}^\intercal$.

To parameterise colour transformations, we use an equivalent representation of colour images in Hue Saturation Value (HSV) space, where each pixel is represented as a tuple $(h, s, v) \in \{[-\pi, \pi] \times [0, 1] \times [0, 1]\}$. Intuitively, HSV space represents the colour of each pixel in a conical space where the hue corresponds to the rotation angle around the cone's vertical axis, the saturation corresponds to the radial distance from the cone's centre, and the value corresponds to the distance along the cone's vertical axis, with a value of 0 corresponding to the tip of the cone, and a value of 1 corresponding to the base of the cone. We colour transform an image by transforming each pixel as

$$\begin{bmatrix} h' \\ s' \\ v' \end{bmatrix} = \begin{bmatrix} (h + 2\pi\eta_h) \mod 2\pi \\ \max(0, \min(s \exp(\eta_s), 1)) \\ \max(0, \min(v \exp(\eta_v), 1)) \end{bmatrix}. \tag{D.19}$$

We therefore obtain $\boldsymbol{\eta} = (\eta_h, \eta_s, \eta_v) \in \{[0, 1] \times \mathbb{R} \times \mathbb{R}\}$. We choose this specific form of parametrising the $\boldsymbol{\eta}$ parameters in order to gain the convenience of simply adding and subtracting in $\boldsymbol{\eta}$ space when carrying out colour transform compositions and inverses. More concretely, with our chosen parametrisation, we obtain the property that $\mathcal{T}_{\boldsymbol{\eta}_1} \circ \mathcal{T}_{\boldsymbol{\eta}_2} = \mathcal{T}_{\boldsymbol{\eta}_1 + \boldsymbol{\eta}_2}$. Therefore, for colour transformations, we can easily perform compositions and inversions in $\boldsymbol{\eta}$ space without resorting to matrix multiplications. In order to achieve this, we first consider hue, which is easy to parametrise in an additive fashion using a modulo operation due to the fact that hue is represented as a rotation angle in HSV space. On the other hand, saturation and value are discontinuous parameters that vary between 0 and 1, and cannot be directly modelled in an additive fashion, as they can't take values outside their range. Instead, we model them as multiplicative factors in $\mathbb{R}^+$, where we

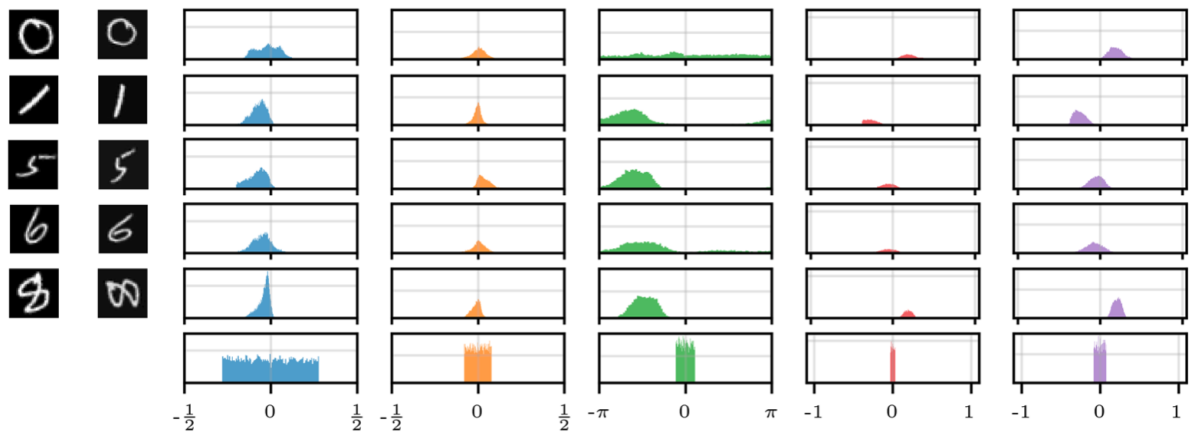


Figure D.3: Learnt augmentation distribution for the MNIST dataset rotated in the range $[-45°, 45°]$ for our SGMs model, and the LieGANs method. The columns correspond to distributions for translation in x, translation in y, rotation, scaling in x, and scaling in y. (Row 1-5) Our SGMs learns accurate ranges of rotational invariance present in the training dataset of a width of $\pi/2$ for most training examples, along with learning the natural invariances present in the training data for translations and scaling. Furthermore, for certain digits (i.e. 0), the SGMs model accurately predicts a uniform distribution from $[-\pi, \pi]$, signifying that rotationally invariant digits such as a 0 would not display a more narrow rotational invariance. **(Row 6)** On the other hand, the LieGANs model learns a single Lie matrix across the entire training dataset that encodes the maximum possible range of transformations, and predicts a uniform distribution between those ranges. It can be seen that LieGANs inaccurately predicts a large range for translations in x, and does not recover the correct range of rotational invariances present in the training dataset.

first exponentiate η_s and η_v to ensure the multiplicative factors are positive. We further clip the obtained values to ensure them in the range $[0, 1]$. This parametrisation allows us to effectively add parameters to compose them, as the multiplicative factors compose in exponent space.

In order to ensure that we can easily backpropagate through the clipping operation, we define a `passthrough_clip` function in Jax, where we define a custom gradient that doesn't zero out gradients even if the inputs to the function are out of bounds. We find that using the `passthrough_clip` operation is essential to training the model.

D.4 Comparisons to LieGAN

In this section, we compare the ability of our method to learn symmetries to LieGANs (Yang et al., 2023), which uses a generator-discriminator framework to automatically discover equivariances from a dataset using generative adversarial training. Similar

to (Yang et al., 2023), we transform the MNIST dataset to have rotations in the range $[-45°, 45°]$, which ensures the dataset contains SE(2) symmetry (rotations and translations). The dataset is processed and our method is trained as described in Section 6.4.1. For LieGANs, following the experimental design of (Yang et al., 2023), we set the number of generator channels to $c = 1$, and consider learnable 6-dimensional Lie matrices in the generator model. The discriminator model consists of a pre-trained LeNet5 feature extractor as the backbone, and the validator is a 3-layer MLP with 512 hidden units and ReLU activations. We train the GANs for 100 epochs with a batch size of 64, and obtain the following Lie matrix

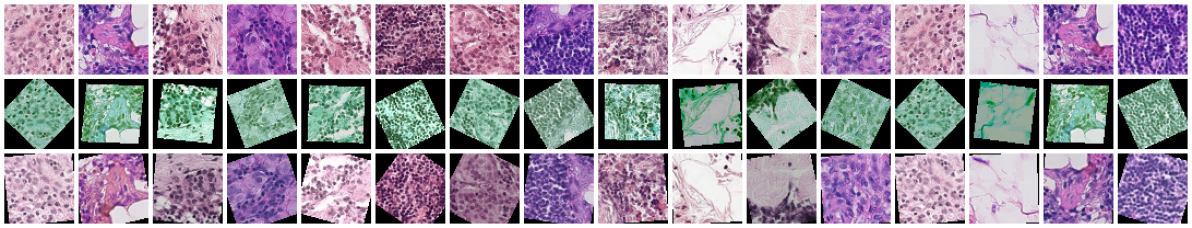
$$L = \begin{bmatrix} 0.02 & -0.34 & 0.28 \\ 0.33 & 0.08 & -0.05 \\ 0 & 0 & 0 \end{bmatrix}.$$

In Figure D.3, we can see that LieGANs struggles to correctly recover the range of invariances present in the training dataset, especially for translations in x. It is also unable to provide a fine-grained representation of invariances depending on specific examples or types of digits. We note that we re-implemented the rotated MNIST experiment from Yang et al. (2023), as the code for the image domain experiments was not open-source. Hence, the choice of using a pre-trained LeNet5 model for the discriminator and the specific hyperparameter configurations were informed decisions made by us based on ablations. However, our results appear to be in line with those presented by Yang et al. (2023); concretely, we note that the results presented in their paper also display a mismatch between the invariances present in the dataset and those learned by LieGANs. For example, in their Figure 11, we see that the sampled digits are often rotated by significantly more than 45°. Furthermore, we see evidence of typical GANs mode collapse, with many very similar rotations for each digit.

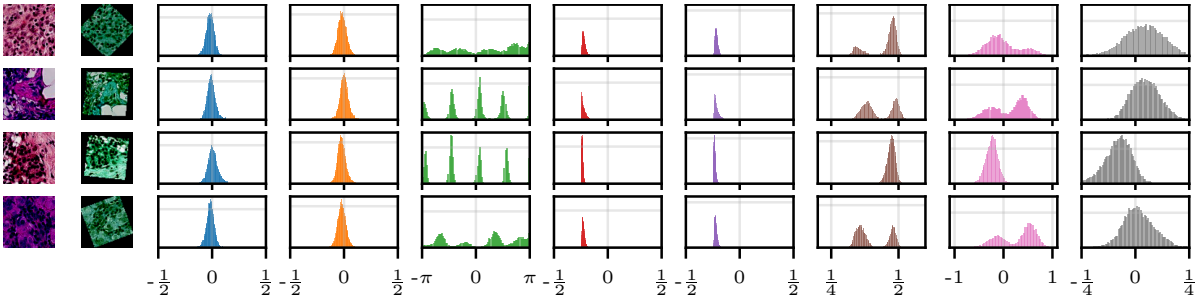
D.5 PatchCamelyon — Boundary Effects

In this section, we provide a "negative" result for our SGMs when applied to the PatchCamelyon dataset (Veeling et al., 2018). The examples in this dataset, unlike those used in Section 6.4, contain "content" up to the boundaries of the images.

Figure D.4 shows examples of the prototypes and learned distributions for this dataset, with affine and colour transformations. We allowed the model to learn any rotations within $\pm 180°$, while the actual dataset has a rotational invariance of $\pm n \times 90°$. We see that in some cases the prototypes are rotated by close to $\pm n \times 45°$ relative to the original



(a) Top: samples from the test set. **Mid:** prototypes for each test example. **Bot:** resampled versions of each test example given the prototype.



(b) From left to right, test examples, their prototypes, and the corresponding marginal distributions over translation in x, translation in y, rotation, scaling in x, scaling in y, hue, saturation, and value.

Figure D.4: Prototypes and learned distributions for PatchCamelyon.

images. In other cases, the rotation of the prototypes relative to the original images is closer to $\pm n \times 90°$. In the latter case, the learned distribution over rotation is close to the true distribution, but in the former case, the model learns a distribution that is closer to uniform. As a result, the resampled digits often display boundary effects that are not present in the original dataset. Otherwise, our SGM has learned reasonable distributions for translation, scaling, and HSV transformations.

Figure D.5 compares a standard VAEs with AugVAEs, an SGMs-VAEs hybrid model. We see that for small amounts of data, the VAEs and AugVAEs perform similarly. However, as the amount of training data increases, the VAEs performs better. This is likely because the SGM has not learned the true distribution over rotations. This "negative" result highlights the importance of correctly choosing the prior transformation distributions in certain settings. In this case, the performance of the SGMs would have been improved by choosing a categorical distribution over rotations.

D.6 Additional Results

In this section, we provide additional plots to supplement those in Section 6.4.

Figure D.6 expands on Figure 6.8b in two ways. Firstly, it makes it clear that our inference network is able to provide the same or very similar prototype for observations

in the same orbit. Secondly, it provides provides many more resampled examples of each digit, further demonstrating that our SGM has correctly captured the symmetries present in the dataset. Figure D.7 expands on Figure 6.8c in the same way.

Figure D.8 provides the learnt marginal distributions for the digits in Figure D.7. Here we manually controlled the distributions over hue and saturation when loading the dataset, so we know that the range of the hue distribution should be approximately π, while the range of the saturation distribution should be around 0.3. We see that this is indeed the case. We did not control the value of the images, so it is more difficult to interpret those. However, given that most (non-black) pixels are bright (i.e., close to 1), it makes sense that our SGM learns multiplicative values closer to 1.

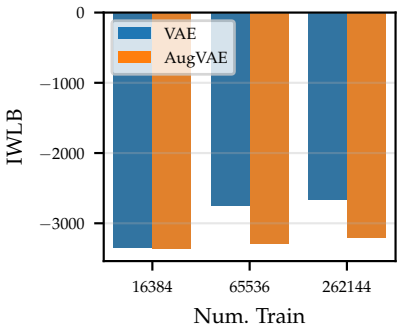


Figure D.5: VAE data-efficiency for PatchCamelyon.

Finally, Figure D.9 extends our dSprites results in two ways. Firstly, it provides many more resampled sprites, which also serve to further demonstrate that our SGM has captured the symmetries correctly. Secondly, the figure includes empirical distributions of positions of each of the classes of digits, which we have carefully controlled as described in Section D.3.3. These empirical distributions for the dataset are compared with empirical distributions for our resampled sprites. We see that although the resampled densities don't match the original densities perfectly, their general shapes and ranges are correct.

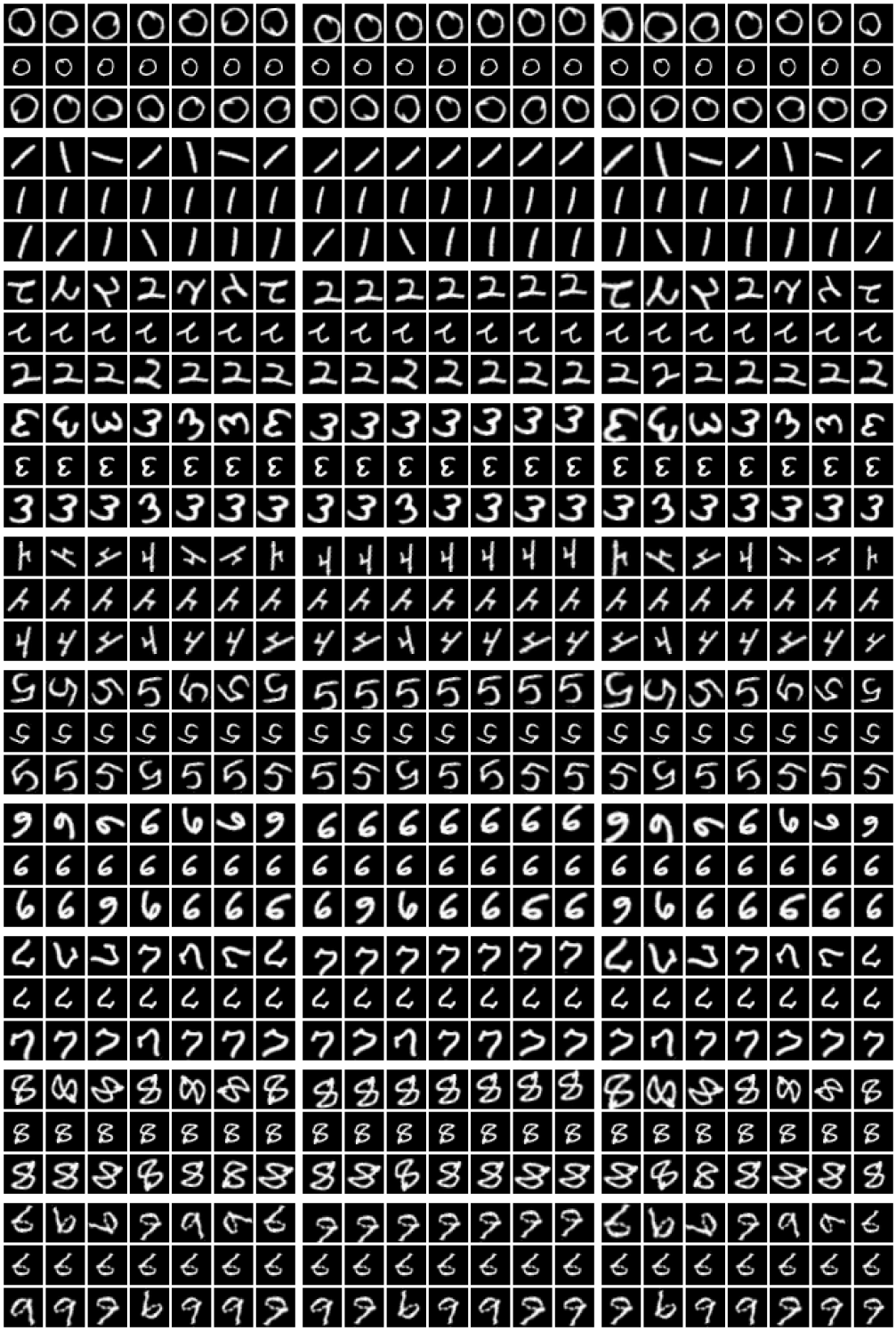


Figure D.6: Columns from left to right: only rotation, only translations, translation + rotation + scaling. Each of the blocks in this figure follows the same format. **Top:** 7 examples from the same orbit. **Mid:** The corresponding prototypes. **Bot:** Resampled versions of the digits, given the prototypes.

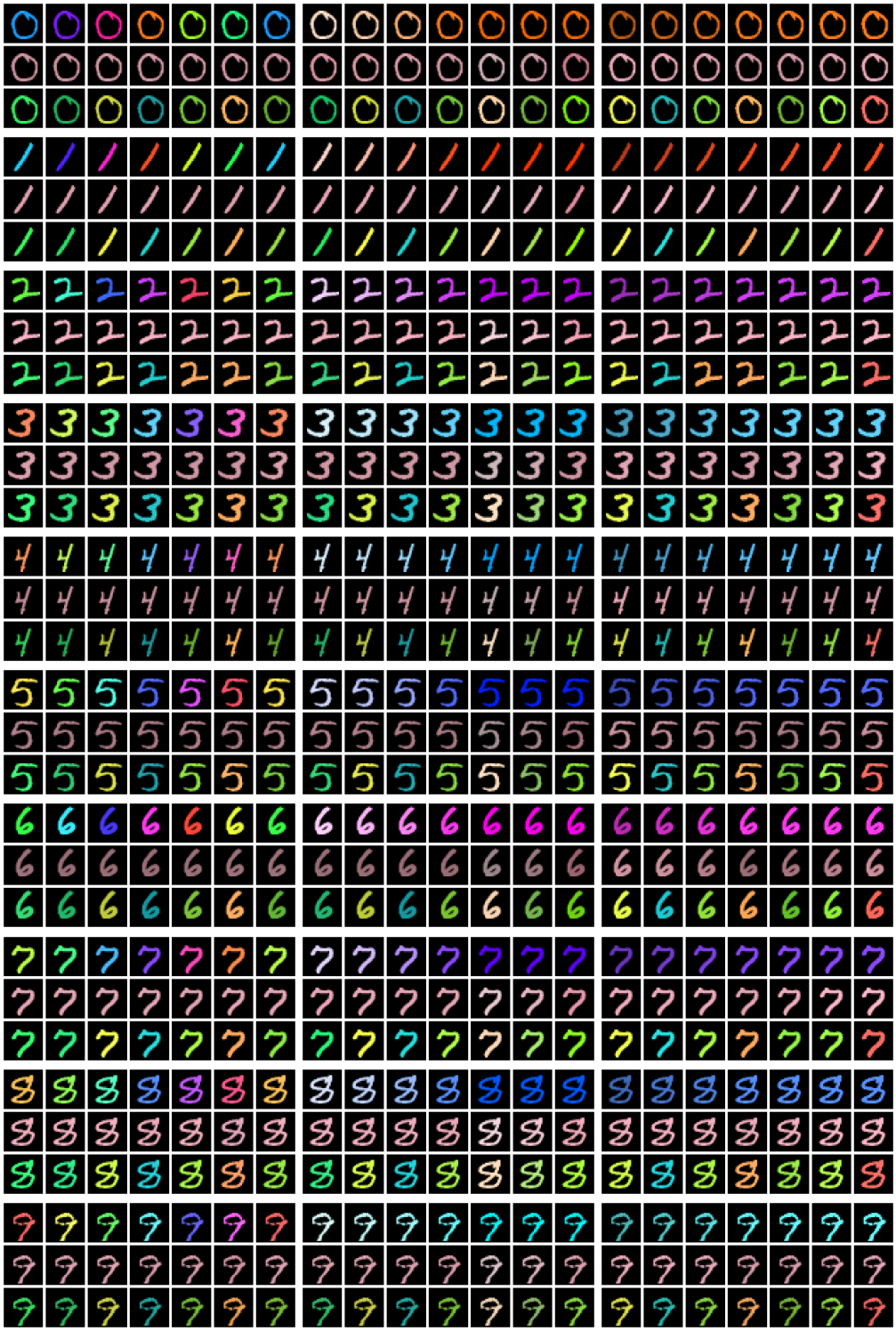


Figure D.7: Columns from left to right: only hue, only saturation, only value. Each of the blocks in this figure follows the same format. **Top:** 7 examples from the same orbit. **Mid:** The corresponding prototypes. **Bot:** Resampled versions of the digits, given the prototypes.

Figure D.8: From left to right, test examples from MNIST with added hue in the range 0 to 0.6π, and saturation scaled by a factor in 0.6 to 0.9, their prototypes, and the corresponding marginal distributions over hue, saturation, and value.

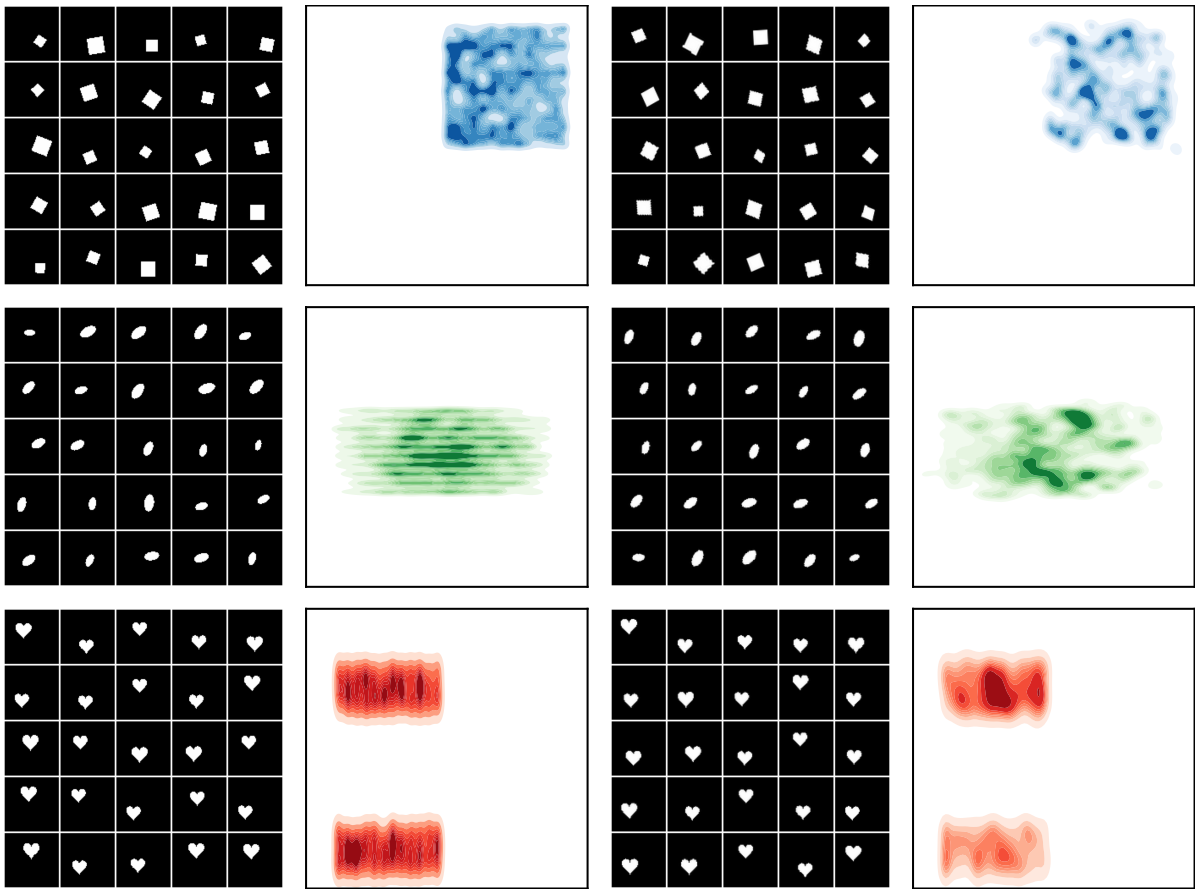


Figure D.9: From left to right, samples from dSprites, the empirical distribution over the positions of the sprites, sprites resampled using our SGM, and the empirical distributions over the resampled sprites' positions. We see that the resampled sprites are visually very similar to the original sprites in terms of sizes, rotations, and positions. Furthermore, we see that the empirical distributions match in terms of ranges, although they are imperfect in densities.