Approximate Bayesian Deep Learning for Resource-Constrained Environments

Meet Prakash Vadera
University of Massachusetts Amherst

APPROXIMATE BAYESIAN DEEP LEARNING FOR RESOURCE-CONSTRAINED ENVIRONMENTS

A Dissertation Presented

by

MEET PRAKASH VADERA

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2022

Manning College of Information and Computer Sciences

APPROXIMATE BAYESIAN DEEP LEARNING FOR RESOURCE-CONSTRAINED ENVIRONMENTS

A Dissertation Presented

by

MEET PRAKASH VADERA

Approved as to style and content by:

ACKNOWLEDGMENTS

As I reflect on the last few years of my life, I feel very lucky to have had many amazing people help me grow. The first person I would like to thank is my advisor, Ben Marlin. Ben is one of the smartest individuals I have ever met. He is thorough and hands-on when it comes to research & teaching, and his work ethic is inspiring. Ben has always encouraged me to explore the problems that I found interesting, and provided valuable guidance when I needed it. The amount of support and patience that Ben has rendered me has been much higher than I had any right to expect. Thank you, Ben, for everything!

I am also very grateful to my thesis committee members - Deepak Ganesan, Subhransu Maji, and Andrew Gordon Wilson. Individually, they are all amazing researchers, and together they have been a terrific thesis committee. Their useful comments and suggestions have helped me improve this thesis. Through my Ph.D. journey, I have also had the opportunity to work with some great external collaborators on different projects. They include Adam Cobb, Jinyang Li, and Brian Jalaian. I would also like to thank my labmates Satya Narayan Shukla, Colin Samplawski, Steve Li, Conrad Holtsclaw and Hui Wei. I've enjoyed working with them on different projects, and have benefitted by brainstorming with them on various occasions. During my Ph.D. program, I have also had the opportunity to participate in internships and have also greatly benefitted through them. I am thankful to the managers and mentors that I had during my internships: Tom Walsh (Kronos), Soumya Ghosh & Kenney Ng (IBM Research), Charles Brett, Amir Salimi, Paul Savastinuk & Evan Welbourne (Amazon).

I've also made some amazing friends through my time here at UMass. The first friend I'd like to thank is Akanksha Atrey. I've enjoyed countless meals with her, confided in her and sought her counsel during tough times, and am also grateful to her husband and her entire family for treating me as their own. I am also super grateful to Deep Chakraborty, Vinita Hissaria, Aarohi Gupta, Ishita Dasgupta, Bhanu Pratap Singh Rawat, Manasa K., Tejas Savalia, Abhyuday Jagannatha, Pavitra Dangati, Dhruvesh Patel, Priyadarshi Rath, Mudit Bhargava, Mohit Surana, Siva Prasad, Nishank Jain, Ankita Mehta, Karma Patel, Jatindeep Singh, and Manasi Wali for their friendship.

I also can't overstate the importance of my family. My parents (Meena & Prakash Vadera) have made a lot of sacrifice in their lives so that my sister and I could get the best possible education without any burden. I would not have been able to do anything in my life without their continuous love, support, and encouragement. My sister, Khyati, has always been an inspiring figure in my life; she has always shown that there is nothing you cannot achieve if you work hard enough. I am also extremely grateful to my brother-in-law Kartik for his unwavering support. Finally, my little nephew Rieshaan also deserves acknowledgment - his smile and innocence has been a great source of happiness for us ever since he's entered our lives.

Funding acknowledgement: Research reported in this thesis was sponsored in part by the CCDC Army Research Laboratory under Cooperative Agreement W911NF-17-2-0196 (ARL IoBT CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

ABSTRACT

APPROXIMATE BAYESIAN DEEP LEARNING FOR RESOURCE-CONSTRAINED ENVIRONMENTS

SEPTEMBER 2022

MEET PRAKASH VADERA

B.Tech., INDIAN INSTITUTE OF TECHNOLOGY GANDHINAGAR

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Benjamin M. Marlin

Deep learning models have shown promising results in areas including computer vision, natural language processing, speech recognition, and more. However, existing point estimation-based training methods for these models may result in predictive uncertainties that are not well calibrated, including the occurrence of confident errors. Approximate Bayesian inference methods can help address these issues in a principled way by accounting for uncertainty in model parameters. However, these methods are computationally expensive both when computing approximations to the parameter posterior and when using an approximate parameter posterior to make predictions. They can also require significantly more storage than point-estimated models.

In this thesis, we address a range of questions related to trade-offs between the quality of inference and prediction and the computational scalability of Bayesian deep learning methods. We begin by developing a framework for comprehensive evaluation of Bayesian neural network models and applying this framework to a range of existing models and inference methods. Second, we address the problem of providing flexible trade-offs between prediction quality, run time, and storage by developing and evaluating a general framework for distilling expectations with respect to the Bayesian posterior distribution of a deep neural network classifier. Third, we investigate the trade-offs between model sparsity and inference performance for deep neural network models using several approaches to deriving sparse model structures. Fourth, we present a framework for correcting approximate posterior predictive distributions, encouraging them to prefer high-utility decisions. Finally, we investigate the use of approximate Bayesian deep learning in object detection and present an evaluation of approaches for quantifying different facets of uncertainty related to object classes and locations.

CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The success of deep learning has been widespread across different areas including computer vision, natural language processing, speech recognition, and more. [Graves et al., 2013, Huang et al., 2016, Devlin et al., 2018, Suzuki, 2017, Minaee et al., 2020]. Key components driving the overall success of deep learning-based methods include advances in learning algorithms, neural network architectures, computing hardware including graphics processing units (GPUs) and tensor processing units (TPUs), and the availability of large labeled data sets.

However, as deep learning models are being deployed in fielded intelligent systems, several challenges have become increasingly prominent. These challenges include the deployment-time occurrence of high confidence errors, the need to be robust to out-of-distribution inputs, and the potential for in-domain adversarial inputs. High confidence errors occur when a probabilistic machine learning model ascribes high probability to an incorrect output (e.g., a class label in a classification setting) [Guo et al., 2017]. Deployed deep learning models can also encounter inputs from data distributions that differ systematically from the distribution they were trained on. In such cases, models need to avoid making high confidence errors. [Hendrycks and Gimpel, 2017]

One of the important factors contributing to susceptibility to these problems is model uncertainty. Supervised deep learning models are most commonly trained by optimizing their parameters to minimize a training loss function. This approach yields a single locally optimal setting of the model parameters, which are then used to make predictions at deployment time. However, given the large number of parameters used in current models, there typically exist multiple qualitatively different sets of parameters yielding similar training loss function values, but making different predictions on future inputs. Common training procedures select a single set of such parameters, despite the fact that for a given amount of training data, there may be significant uncertainty over what the best model parameters actually are.

Bayesian inference provides a different perspective on the problem of training deep neural network models that attempts to represent model uncertainty and propagate it to the point of issuing predictions and making decisions. In Bayesian inference, the unknown model parameters are formally treated as random variables. The goal becomes to infer the posterior probability distribution over the unknown model parameters given the available data. When the data support multiple distinct interpretations in terms of settings of the model parameters, the model posterior will reflect this uncertainty. Incorporating model uncertainty into prediction and decision-making typically decreases overconfidence in predictions via the Bayesian model averaging effect, and can also increase robustness to out-of-distribution and adversarial inputs [Wilson, 2020].

Although Bayesian inference techniques appear promising, there are certain key challenges that remain with implementing Bayesian neural networks. The first challenge is scalability. Bayesian neural networks are computationally expensive, as the posterior predictive distribution is intractable, and we have to resort to approximation. This makes the methods prohibitive at test time. The second challenge is inference quality. Bayesian neural network posteriors are highly multimodal, and thus it's important for Bayesian inference algorithms to return a diverse representation of the posterior distribution, within a reasonable amount of training time. Furthermore, if there exists a utility function used for downstream decision-making, our approximate inference algorithms need to quickly adapt to them for effective deployment. Finally, as the area of Bayesian neural networks sees rapid growth, it is important to establish common benchmarks based on comprehensive evaluation principles that reflect multiple aspects of performance, robustness and scalability.

In this dissertation, we aim to tackle these challenges by developing Bayesian deep learning algorithms that are suited for practical deployments. For practical applications, we need algorithms that can scale well in constrained environments and also have lower prediction latency. Furthermore, for constantly changing utility functions, our methods should be able to adapt to them quickly. In this dissertation, we explicitly address each of these requirements. In the next section, we provide an outline for the different chapters and summarize the contribution of each chapter.

1.1 Dissertation Outline

In **Chapter 2**, we review the literature in the area of Bayesian deep learning. We provide an overview of different approaches and a discussion around how they trade-off scalability-performance against each other. We provide an overview and discussion of model pruning and distillation approaches to tackle the test-time scalability of Bayesian neural networks. Finally, we also briefly describe Bayesian decision theory, which is an elegant framework for decision-making under uncertainty, given a downstream utility function. This is an important framework, as often for practical applications, the utility of different decisions based on inputs is asymmetric, and thus we need to optimize for maximizing utility on decisions.

In **Chapter 3**, we present a benchmarking framework consisting of different Bayesian neural network models, data sets, and downstream tasks. An important piece of our benchmark is the set of evaluation principles. These principles are aimed at evaluating the characteristics of model calibration, robustness to out-of-distribution inputs, aside from just accuracy. This helps us establish a rigorous benchmark, which is also useful for the wider community. We call this benchmark *URSABench* (the Uncertainty, Robustness, Scalability, and Accuracy Benchmark).

Next, in **Chapter 4**, we focus on the issue of scalability of Bayesian neural networks at test time. We present a general framework for distilling expectations with respect to the Bayesian posterior distribution of a deep neural network classifier, extending prior work on the Bayesian Dark Knowledge framework. We call this framework the *Generalized Bayesian Posterior Expectation Distillation (GPED)* framework. The proposed framework takes as input "teacher" and "student" model architectures and a general posterior expectation of interest. The distillation method performs an online compression of the selected posterior expectation using iteratively generated Monte Carlo samples. We focus on the posterior predictive distribution and expected entropy as distillation targets. We investigate several aspects of this framework, including the impact of uncertainty and the choice of student model architecture. We study methods for student model architecture search from a speed-storage-accuracy perspective and evaluate down-stream tasks leveraging entropy distillation including uncertainty ranking and out-of-distribution detection.

In **Chapter 5**, we investigate the potential of sparse network structures to flexibly trade-off model storage costs against predictive performance and uncertainty quantification ability. We also investigate the impact of sparsity on mixing. We use stochastic gradient MCMC methods as the core Bayesian inference method and consider a variety of approaches for selecting sparse network structures, including random and iterative pruning-based methods. Our results show that both approaches can provide useful trade-offs between posterior predictive performance and storage costs.

In **Chapter 6**, we turn our focus to the area of Bayesian decision theory. Bayesian decision theory provides an elegant framework for acting optimally under uncertainty when tractable posterior distributions are available. However, for most cases, and certainly for deep neural networks, computing the posterior distribution and the posterior predictive distribution is not tractable. In this work, we develop methods to correct approximate posterior predictive distributions, encouraging them to prefer high-utility decisions. In contrast to previous work, our approach is agnostic to the choice of the approximate inference algorithm, allows for efficient test time decision-making through amortization, and empirically produces higher quality decisions. We demonstrate the effectiveness of our approach through controlled experiments spanning a diversity of tasks and datasets.

In **Chapter 7**, we consider the application of approximate Bayesian inference to the task of object detection using detection transformers. Object detection is a large-scale practical application in deep learning, making it an interesting problem to explore from the viewpoint of uncertainty quantification. As most of the past work in this area has looked at the "correctness" aspect of the object detectors, we outline a set of evaluation principles to evaluate the probabilistic nature of model predictions. These evaluation principles look at different facets of uncertainty, such as location uncertainty, class uncertainty, and objectness uncertainty. Through our controlled experiments, we evaluate different inference methods against the evaluation principles outlined, examine their runtime latency breakdown, and also provide latency-aware performance comparison.

Finally, in **Chapter 8**, we provide conclusions, and a set of future directions that build upon this thesis.

Bibliographical note: **Chapter 2** is adapted from Vadera and Marlin [2021]. **Chapter 3** is adapted from Vadera et al. [2022]. **Chapter 4** is adapted from Vadera et al. [2020b]. **Chapter 6** is adapted from Vadera et al. [2021].

CHAPTER 2

BACKGROUND AND RELATED WORK

Approximate Bayesian deep learning methods hold significant promise for addressing several issues that occur when deploying deep learning components in intelligent systems, including mitigating the occurrence of over-confident errors and providing enhanced robustness to out of distribution examples. In this chapter, we present a range of approximate Bayesian inference methods for supervised deep learning and highlight the challenges and opportunities when applying these methods. We highlight several potential solutions to decreasing model storage requirements and improving computational scalability, including model pruning and distillation methods.

The remainder of this chapter is organized as follows. In Section 2.1 we begin by providing a comprehensive discussion of Bayesian supervised learning, approximate Bayesian inference, and the scalability challenges of deploying current Bayesian deep learning model representations. Next, in sections 2.2 and 2.3 we discuss model compression techniques that can be leveraged for compressing Bayesian posterior distributions. These approaches either look at compressing each member of the model ensemble, or compress the entire ensemble into a surrogate model.

Bibliographical note: This chapter is adapted from Vadera and Marlin [2021].

2.1 Bayesian Deep Learning and the Challenge of Scalability

In this section, we introduce the fundamental concepts of Bayesian inference for supervised deep learning along with foundational approximation methods. We discuss the scalability challenges when deploying such methods.

2.1.1 Bayesian Supervised Learning

Supervised learning forms the core of machine learning-based prediction systems. In a supervised learning problem, we are given a dataset $\mathcal{D}$ consisting of input-output pairs $\{(\mathbf{x}_i, y_i) | 1 \leq i \leq N\}$, where $\mathbf{x}_i \in \mathbb{R}^D$ is the input or feature vector and $y_i \in \mathcal{Y}$ is the output or prediction target. We let $\mathcal{D}^x$ be the dataset of inputs and $\mathcal{D}^y$ be the dataset of outputs. The nature of $\mathcal{Y}$ depends on the task at hand. For classification tasks, $\mathcal{Y}$ is a finite set, whereas for regression tasks, we usually have $\mathcal{Y} \in \mathbb{R}$. There also exist tasks, such as object detection, that are a combination of classification and regression tasks. In this chapter, we specifically focus on the classification setting in supervised learning, where the goal is to learn a function $f : \mathbb{R}^D \rightarrow \mathcal{Y}$ that can accurately predict the outputs from the inputs.

In probabilistic supervised learning, we construct the prediction function using a conditional probability model of the form $f_\theta(\mathbf{x}) = p(y|\mathbf{x}, \theta)$ where $\theta \in \mathbb{R}^K$ are the model parameters. The conditional likelihood of the inputs given the outputs and the parameters is denoted by $p(\mathcal{D}^y|\mathcal{D}^x, \theta)$. Under the assumption that the outputs are independent and identically distributed given their corresponding inputs, we have that $p(\mathcal{D}^y|\mathcal{D}^x, \theta) = \prod_{i=1}^N p(y_i|\mathbf{x}_i, \theta)$. A key ingredient in Bayesian inference, as well as in traditional point-estimated neural networks, is the prior distribution $p(\theta|\lambda)$ over model parameters. As the name suggests, the prior distribution represents our beliefs about the distribution of the model parameters prior to analyzing the data. The prior distribution can have its own hyperparameters, here denoted by λ [Neal, 1996].

Bayesian inference involves the computation of the posterior distribution over the unknown model parameters given a training dataset $\mathcal{D}_{tr}$ and the prior. The parameter posterior is obtained using Bayes theorem, as shown in Equation equation 2.1.

$$p(\theta|\mathcal{D}_{tr}, \lambda) = \frac{p(\mathcal{D}^y_{tr}|\mathcal{D}^x_{tr}, \theta)p(\theta|\lambda)}{\int p(\mathcal{D}^y_{tr}|\mathcal{D}^x_{tr}, \theta)p(\theta|\lambda)d\theta} \tag{2.1}$$

The denominator in the parameter posterior (referred to as the "evidence" term) is intractable to compute for most ML models, including for deep neural networks [Neal, 1996]. As a result, the computation of the exact posterior distribution is intractable. However, in practice, the quantity of interest is often not the parameter posterior distribution itself, but rather low-dimensional expectations under the parameter posterior.

One key posterior expectation in the supervised learning setting is the posterior predictive distribution, which is necessary and sufficient for making maximum probability predictions for outputs given inputs while integrating over the uncertainty in the model parameters. The posterior predictive distribution computation is shown in Equation equation 2.2.

$$p(y|\mathbf{x}, \mathcal{D}_{tr}, \lambda) = \mathbb{E}_{p(\theta|\mathcal{D}_{tr},\lambda)}[p(y|\mathbf{x}, \theta)] \tag{2.2}$$

Another posterior expectation that is useful in uncertainty quantification is the expected posterior predictive entropy. Posterior predictive entropy (also referred to as the *total uncertainty* of the predictive distribution) can be decomposed into quantities referred to as *expected data uncertainty* and *knowledge uncertainty* [Depeweg et al., 2017]. These three forms of uncertainty are related by the equation shown below:

$$\begin{aligned}\underbrace{\mathcal{H}\left[\mathbb{E}_{p(\theta|\mathcal{D})}\left[p\left(y|\boldsymbol{x},\theta\right)\right]\right]}_{\text{Total Uncertainty}} = & \underbrace{\mathcal{I}\left[y,\theta|\boldsymbol{x},\mathcal{D}\right]}_{\text{Knowledge Uncertainty}} \\ & + \underbrace{\mathbb{E}_{p(\theta|\mathcal{D})}\left[\mathcal{H}\left[p\left(y|\boldsymbol{x},\theta\right)\right]\right]}_{\text{Expected Data Uncertainty}} \end{aligned} \tag{2.3}$$

Knowledge uncertainty can be efficiently computed as the difference between total uncertainty and expected data uncertainty, both of which are functions of posterior expectations. Recent work has leveraged these uncertainty estimates to explore a range of down-stream tasks such out-of-distribution detection, misclassification detection, and active learning that rely on uncertainty quantification and decomposition.[Wang et al., 2018, Malinin et al., 2020, Houlsby et al., 2011, Holtsclaw et al., 2019, Kirsch et al., 2019]. However, all of these posterior expectations are also intractable to compute exactly for deep learning models. We thus next turn to the problem of approximate Bayesian inference methods.

2.1.2 Approximate Bayesian Inference for Supervised Learning

As indicated in the previous subsection, posterior expectations including the posterior predictive distribution needed for Bayesian supervised learning is intractable in its original form. To tackle this problem, there is a significant body of work in the area of approximate Bayesian inference. The ultimate goal of these approximation methods is to compute approximate posterior expectations that are close to their theoretical counterparts. Approximate Bayesian methods can be broadly divided into three categories: Markov Chain Monte Carlo (MCMC) methods, surrogate density estimation methods, and other approximation methods. We describe each category of methods below and discuss their deployment challenges.

2.1.2.1 Markov Chain Monte Carlo Methods

MCMC methods provide an approximation to the intractable parameter posterior $p(\theta|\mathcal{D}, \lambda)$ via a set of samples drawn for this distribution. MCMC methods simulate a Markov chain that converges to the parameter posterior as its steady state distribution. The simulated states of the Markov chain after convergence correspond to samples from the parameter posterior. Once we collect a set of parameter samples of the desired size, we can approximate expectations with respect to the parameter posterior using empirical averages over the sampled parameter values [Smith and Roberts, 1993]. For example, the posterior predictive distribution can be approximated as shown in Equation equation 2.4. As we can see, computing the Monte Carlo approximation to the posterior predictive distribution is very similar to computing the predictive distribution of a model ensemble.

$$\begin{aligned} p(y|\mathbf{x}, \mathcal{D}, \lambda) &= \mathbb{E}_{p(\theta|\mathcal{D},\lambda)}[p(y|\mathbf{x}, \theta)] \\ &\approx \mathbb{E}_{p_{\mathrm{MC}}(\theta|\mathcal{D},\lambda)}[p(y|\mathbf{x}, \theta)] \\ &= \frac{1}{S}\sum_{s=1}^{S} p(y|\mathbf{x}, \theta_s) \end{aligned} \tag{2.4}$$

Examples of classical MCMC methods include the Gibbs sampler [Casella and George, 1992] and the Metropolis-Hastings sampler [Chib and Greenberg, 1995]. The earliest work on MCMC samplers for neural networks traces back to the application of Hamiltonian Monte Carlo methods [Duane et al., 1987]. While a number of MCMC methods have since been developed with improved properties including slice sampling [Neal, 2003], elliptical slice sampling [Murray et al., 2010], and Riemann manifold methods [Girolami and Calderhead, 2011], these methods all require using all the available data when computing the likelihood term needed for posterior inference. Although only linear in the number of data cases, this can be a highly expensive operation for large data sets and models and can render MCMC methods practically infeasible in scenarios where stochastic gradient descent (SGD) [Bottou, 2010] can be usefully applied to optimize model parameters.

However, recent advances in MCMC approaches have enabled the use of SGD-like mini-batch algorithms, greatly extending the range of applicability of MCMC methods. Prominent examples of such approaches include stochastic gradient Langevin dynamics (SGLD) [Welling and Teh, 2011], stochastic gradient Hamiltonian Monte Carlo (SGHMC) [Chen et al., 2014b] and their cyclic learning rate versions as presented in [Zhang et al., 2020]. SGHMC algorithm involves computing the potential function $\tilde{U}(\theta)$ on a minibatch $\mathcal{B}$ as shown in the equation below, followed by running the next two update equations:

$$\tilde{U}(\theta) = -\log(p(\mathcal{B}|\theta)) - \log(p(\theta|\lambda)) \tag{2.5}$$

$$\theta_k = \theta_{k-1} + v_{k-1} \tag{2.6}$$

$$v_k = v_{k-1} - \alpha_k \nabla\tilde{U} - \eta v_{k-1} + \sqrt{2(\eta - \hat{\gamma})\alpha_k}\epsilon_k \tag{2.7}$$

In the above equations k denotes the iteration, v_k denotes the momentum term, $(1-\eta)$ denotes the momentum factor, ϵ_k is drawn from an identity Gaussian distribution, $\nabla\tilde{U}$ denotes the gradient approximation obtained using a minibatch, $\hat{\gamma}$ and α_k denotes the instantaneous step size. The process highlighted in the above equations is run iteratively to draw new samples from the posterior. For practical purposes, most implementations set $\hat{\gamma}$ to 0 [Zhang et al., 2020]. Interestingly, SGLD can be derived from SGHMC by setting the momentum factor to 0.

Another important property that determines the efficacy of MCMC methods is the degree of mixing. The degree of mixing refers to how efficiently the Markov chain traverses the posterior distribution after convergence [Brooks et al., 2011]. Better mixing enables faster collection of a more diverse set of parameter samples. However, the mixing properties of MCMC methods depend on the dimensionality of the parameter space. Modern deep neural networks can have an extremely large number of parameters (millions or more), potentially leading to inadequate mixing.

An alternative to sampling in the original parameter space $\mathbb{R}^K$ is to sample in a reduced dimensional space $\mathbb{R}^{K'}$ for $K' \ll K$ and to project back to the full-dimensional space. This process is termed *subspace inference* [Izmailov et al., 2019]. Subspaces can be generated using singular value decomposition (SVD) applied to SGD iterates, or by generating random projection matrices, among other possible options [Izmailov et al., 2019]. It is worth noting that these methods introduce bias by restricting the sampler to operate in a subspace of $\mathbb{R}^K$.

2.1.2.2 Surrogate Density Methods

An alternate approach to MCMC methods is approximating the original posterior distribution via an analytically tractable parameterized surrogate distribution. Thus, given the original posterior distribution $p(\theta|\mathcal{D}_{tr}, \lambda)$, surrogate density methods aim to approximate the true posterior using an auxiliary distribution $q(\theta|\phi)$, where ϕ are the auxiliary parameters [Jordan et al., 1999, Jaakkola and Jordan, 2000, Ghosh et al., 2016, Minka, 2001]. A common approximation is the use of a multivariate Gaussian distribution with a diagonal covariance matrix $\mathcal{N}(\theta; \mu, \Sigma)$, also known as mean-field variational inference. Here, the auxiliary parameters are $\phi = [\mu, \Sigma]$. The main reason why mean-field variational inference is popular is due to the simple "re-parameterization trick" that makes sampling from the auxiliary distribution straightforward [Blundell et al., 2015]. With the re-parameterization trick, we can sample $\theta_k \sim \mathcal{N}(\mu_k, \Sigma_{kk})$ by first drawing $\eta \sim \mathcal{N}(0, 1)$, followed by the linear transformation $\theta_k = \mu_k + \eta \cdot \sqrt{\Sigma_{kk}}$. This allows us to backpropagate through the variational parameters while drawing samples of the model parameters to approximate the objective functions used for learning.

Now, for estimating the auxiliary parameters, we need an objective function that can measure the discrepancy between the surrogate distribution and the ground truth posterior. Needless to say, this objective function must also be computationally tractable. The most commonly used discrepancy function is the Kullback-Leibler (KL) divergence as shown below [MacKay, 2003].

$$\mathrm{KL}(p(\theta)||q(\theta)) = \mathbb{E}_{p(\theta)}\left[\log\left(\frac{p(\theta)}{q(\theta)}\right)\right]$$

The KL divergence is a directional divergence, and thus is not symmetric in its arguments. This results in two different measures depending on the directionality. When the surrogate posterior is used as the first argument, the result is the variational inference (VI) framework [Jordan et al., 1999, Jaakkola and Jordan, 2000]. When the surrogate posterior is used as the second argument, the result is the expectation propagation (EP) framework [Minka, 2001]. This often leads to VI methods having mode seeking behavior, as they are not forced to match the support of the original posterior. EP methods on the other hand are forced to match the support, but this can often lead to incorrect mode estimation as it must cover the support of the original posterior. These two extremes can also be interpolated by more generalized divergence measures, including alpha divergence [Li and Turner, 2016].

These methods also suffer from scalability issues due to the need to compute the log likelihood and its gradient over the entire dataset. Similar to the stochastic gradient version of MCMC methods introduced earlier, advances in the past decade have led to more scalable methods in this family, including stochastic variational inference [Hoffman et al., 2013], that are able to accommodate large-scale datasets using mini-batch gradients.

In addition to mean field VI, other more advanced approximations are possible, such as multiplicative normalizing flows (MNF) [Louizos and Welling, 2017], Bayesian hypernetworks [Krueger et al., 2017], and Rank-1 factorization [Dusenberry et al., 2020]. In multiplicative normalizing flows, we choose a simple density function such as the isotropic Gaussian distribution, and use a bijective function to transform the samples drawn from the simple density function to form more complex distributions. The Bayesian hypernetworks approach builds on the MNF approach and uses a neural network model to transform the samples drawn from a simpler density function to model a more complex posterior distribution. Finally, the rank-1 factorization approach represents the model parameters as a product of two rank-1 factors thereby reducing the dimensionality of the base distribution of the approximate posterior. For example, if we have a parameter matrix $\mathcal{W} \in \mathbb{R}^{M\times N}$, this can be re-written as a matrix product of $W_1 \in \mathbb{R}^{M\times 1}$ and $W_2 \in \mathbb{R}^{1\times N}$. This effectively reduces the number of parameters from $\mathcal{O}(M \cdot N)$ to $\mathcal{O}(M + N)$.

MC Dropout is a particularly interesting approach, that is equivalent to approximate variational inference under specific assumptions [Gal and Ghahramani, 2016]. Dropout itself was first introduced as a regularization technique where during every training iteration a pre-determined proportion of activations is randomly set to zero to reduce overfitting. At test time, dropout is switched off and all units participate in making predictions [Srivastava et al., 2014]. In MC Dropout, by contrast, dropout is used at prediction time. This leads to a stochastic forward pass through the point-estimated model. Multiple forward passes through the model are used, and the predictive distributions are averaged. This procedure is equivalent to sampling from a specific approximate variational posterior, but has the advantage that it is very easy to implement.

The major drawback of surrogate density methods is that they introduce bias into the estimation of the posterior distribution unless the true posterior belongs to the family of auxiliary distributions. The degree of bias will depend on the functional form of the auxiliary distribution, and the divergence measure used to estimate the parameters of the auxiliary distribution.

In addition, under these methods, we still face hurdles in computing posterior expectations including the approximate posterior predictive distribution. In particular, even if $q(\theta|\phi)$ is a simple parametric distribution, the expectation $\mathbb{E}_{q(\theta|\phi)}[p(y|\mathbf{x}, \theta)]$ still usually cannot be computed analytically due to the non-linearity of $p(y|\mathbf{x}, \theta)$. As a result, we have to again resort to Monte Carlo approximation, and draw samples from the approximate variational posterior. Generally, these methods trade-off the potential bias in the surrogate parameter posterior for the ability to draw independent samples once the surrogate posterior parameters have been estimated.

2.1.2.3 Additional Approximate Bayesian Inference Methods

With the emergence of Generative Adversarial Networks (GANs) [Goodfellow et al., 2014], there has been work on learning implicit generative model representations of the parameter posterior that can be used at test time to draw an arbitrary but finite number of samples from this posterior approximation. The existing work in this area involves training GANs that can approximate the posterior distribution asserted by SGLD [Wang et al., 2018, Henning et al., 2018]. To compute the approximate posterior predictive distribution, we yet again would use Monte Carlo approximation as shown in the previous subsections. An advantage of these methods is that in theory we do not need to store posterior samples for use during inference, and we can also determine the number of posterior samples to use on the fly.

Deep Ensembles [Lakshminarayanan et al., 2017] have also shown strong performance in terms of representing predictive distributions, and can also be considered as an approximation to Bayesian inference [Wilson and Izmailov, 2020]. However, deep ensembles can be expensive during training, as we need to train each model in the ensemble from scratch. To alleviate this issue, snapshot ensembles [Huang et al., 2017] uses a cyclical learning rate schedule to generate more ensembles in less training time. However, these methods learn a mixture of posterior modes, which is different from an approximation to the full posterior. Regardless, deep ensembles have been shown to yield better performance relative to using small numbers of samples in a traditional MCMC approach. Finally, while comparing the predictive distributions between different approximate inference methods and deep ensembles, Izmailov et al. [2021b] observed that while deep ensembles can achieve better performance in terms of good generalization accuracy and calibration, the predictive distribution was different from the predictive distribution generated using HMC.

2.2 Model Pruning Approaches to Improving the Scalability of Bayesian Deep Learning

As described in the previous section, the storage and computational scalability properties of Bayesian inference for neural network models can be a significant barrier to deployment, despite their potential benefits in the context of intelligent systems. In this section, we discuss model pruning and sparsification approaches that aim to reduce the storage and computation requirement for Bayesian ensembles. These approaches can be broadly divided into unstructured and structured pruning approaches, as we describe below.

2.2.1 Unstructured Pruning

Optimization-based unstructured pruning methods aim to compress neural network models by sparsifying their weight matrices. The earliest work on unstructured neural network pruning dates back to the Optimal Brain Damage method [LeCun et al., 1989]. In the optimal brain damage approach, the authors presented a Taylor series approximation of the objective function, and show that under the assumption that the Hessian matrix is diagonal, the weights corresponding to the second order derivatives that go to zero can be removed with little to no loss in performance. The follow-up Optimal Brain Surgeon method [Hassibi et al., 1993] highlights that the diagonal Hessian assumption can be limiting, leading to the removal of connections that should be retained. It uses second order derivatives to decide which connections to remove, and obtains better generalization on held-out test data compared to the Optimal Brain Damage approach.

Another line of early work in unstructured pruning removes parameters based on their magnitudes. The earliest work in this area dates back to [Hertz et al., 1991]. Since then, magnitude-based unstructured pruning has been revisited in more detail, and it has been found that iterative pruning with fine-tuning can help prune more parameters of a neural network while retaining better predictive performance compared to one-time post-hoc pruning [Han et al., 2015].

In the iterative pruning with fine-tuning approach, we define a pruning rate p (the percentage of weights to prune) for each pruning cycle and remove all weights with magnitudes in the bottom p percentile. We then fine tune the model by optimizing the unpruned weights to a desired level of convergence. We repeat these pruning and fine-tuning cycles until we achieve the desired overall sparsity level.

Experimental results on iterative pruning with fine-tuning have shown that for the ImageNet data set [Deng et al., 2009], the authors were able to reduce the number of active parameters in the AlexNet model [Krizhevsky et al., 2012] by 9 times, and reduce the number of active parameters in VGG16 [Simonyan and Zisserman, 2014] by 16 times. It is important to note that regularization methods (such as the application of an ℓ_2 or ℓ_1 penalty) can help to ensure that the magnitude of the weights that are not contributing to predictive performance are driven to zero. Building upon this, there has been work on further compressing the model parameters using quantization and Huffman coding [Han et al., 2016a] after pruning. There has also been additional work on weight magnitude threshold-based pruning that allows for restoring connections [Guo et al., 2016, Jin et al., 2016, Han et al., 2016b].

The Lottery Ticket Hypothesis [Frankle and Carbin, 2018] proposed an iterative pruning method that is very similar to the method of [Han et al., 2015], which we refer to as Iterative Pruning with Rewinding. This approach differs from basic iterative pruning in that following each pruning iteration, the weights used to initialize the next iteration of the algorithm are formed by combining the pruned weights with the original random weight vector (generated during initialization), instead of the weight vector obtained at the end of the previous pruning iteration. Effectively, the active weights are rewound to their initial values at the start of each round of iterative pruning. Through this approach, the authors found that there are sparse substructures within deep neural networks that, when initialized randomly, achieve very similar performance to the original dense networks with no pruning.

While unstructured pruning methods are well-studied in the context of optimization-based deep learning, they have not received as much attention in the Bayesian deep learning literature despite their potential to reduce the storage complexity of posterior model ensembles. Several applications of unstructured sparsity are possible. First, MCMC methods can be used to generate a posterior ensemble consisting of a set of models. A single round of pruning can then be applied to each of the models to remove a specified percentage of the smallest weights, resulting in an ensemble of weight-sparse models. However, optimization-based fine-tuning can not be applied in this setting without the potential for significantly altering the distribution that the ensemble represents. Importantly, there is potential for the sparsified models in such an ensemble to tolerate much higher levels of sparsity than an individual model due to the averaging that occurs over the elements of the ensemble when making predictions.

An alternative approach is to use optimization-based iterative pruning and fine-tuning to derive a sparse network structure and a starting set of parameter values. MCMC methods can then be initialized to sample within this sparse structure, starting from the parameter values found using optimization. Optimization-based iterative pruning approaches can also potentially be combined with variational Bayesian deep learning methods. In the case of the classical Gaussian mean field approximation, both weights that are close to zero and weights that have high posterior variance could potentially be pruned from the model. Since variational inference is fundamentally an optimization-based procedure, it can also be composed with an iterative pruning and fine-tuning process to more closely mimic the process that is used in standard optimization-based deep learning, as described above.

We note that while unstructured pruning has been proven to preserve prediction accuracy even at high levels of weight sparsity for large optimization-based models, whether and how these savings convert into practical savings in deployed systems is fairly complicated. We first consider the storage properties of weight-sparse models. In particular, the parameter matrices for a weight-sparse model must be stored in a compressed format to yield any storage benefit.

Sparse matrices are typically stored either in *coordinate list* (COO) format or *compressed sparse column/row* (CSC/CSR) format. In these formats, at a high level, we store the indices and values of the non-zero elements of the matrix. For the COO format, for example, the space complexity of the resulting data structure is $\mathcal{O}(3N)$, where N is the number of non-zero elements. Importantly, how we compose weight sparsity with Bayesian deep learning can have significant impact on the storage complexity of the resulting models.

In the first approach described above, we considered independently sparsifying each element of the posterior ensemble. If we retain a total of N non-zero weights for each of the S models, the total storage required is $\mathcal{O}(3NS)$. In the second approach, we considered sampling within a fixed sparse structure. If this sparse structure has N non-zero weights, the total required storage is theoretically, $\mathcal{O}(2N + NS)$ since we only need to store the indices of the non-zero elements once. For $S \gg 2$, this reduces the required storage of the sparse ensemble by 66% over standard COO format. We note that no current sparse matrix formats exist that support this optimization.

Lastly, we note that realizing the computational savings of weight-sparse networks with GPUs and TPUs is also currently a challenge. PyTorch, for example, has sparse matrix support that is currently in beta testing. Existing libraries for edge GPU/TPU-accelerated architectures such as TensorRT currently only support highly restricted sparsity patterns. As a result, fully realizing the theoretical computational benefit of weight-sparse Bayesian deep learning approaches on edge platforms will require additional support for GPU/TPU-accelerated linear algebra operations over sparse matrices.

2.2.2 Structured Pruning

Optimization-based structured pruning methods apply similar pruning principles to those of unstructured pruning, but with the aim of pruning larger structural elements such as entire hidden unit or entire convolutional kernels. The simplest structured pruning methods extend the iterative pruning approaches described above to also include removing hidden units or convolutional kernels that have no incoming connections [Han et al., 2015]. However, the induced sparsity patterns can be such that few units are actually pruned. These basic approaches can be improved by modifying the pruning criteria to prune units where the norm of their incoming weights is in the bottom p percentile across all active units [Li et al., 2016]. This approach ensures that a desired number of units is pruned from the network on each round.

Another set of approaches leverage group LASSO (the least absolute shrinkage and selection operator) regularization to encourage a group of incoming weights to go to zero simultaneously [Zhang and Ou, 2018, Alvarez and Salzmann, 2016, Wen et al., 2016, He et al., 2017]. To apply this approach, we must first must partition the model parameters in the parameter vector θ into K groups $\mathcal{G}_k$. The form of the regularizer is $R(\theta) = \sum_{k=1}^{K} \left(\sum_{j \in \mathcal{G}_k} \theta_j^2 \right)^{1/2}$ when using the group ℓ_1/ℓ_2 regularizer. For a feedforward model, we place all the incoming weights for each hidden unit into a group. Similarly, we collect all the incoming weights for a particular channel in a convolution layer together into a group. The regularizer will then tend to encourage all the weights in a group to go to zero or most of the weights in a group to be non-zero. This can make it much easier to identify structures for pruning using weight norm-based criteria and can require less fine-tuning as most inputs to a unit will tend to already be close to zero prior to pruning.

Structured pruning can also be composed with approximate Bayesian deep learning methods in multiple ways. As with unstructured pruning, we can also separately apply structured pruning to each element of a posterior ensemble. However, the chances that all incoming weights for a unit will be close to zero is many times smaller than the chance that a single weight will be small. In this case, the use of an explicit sparsity inducing prior, such as a spike-and-slab prior, would very likely be necessary to obtain meaningful pruning. The approach of using optimization-based structured pruning methods to select a structure and an initial set of weights to run MCMC-based methods from is also applicable and is a potentially promising approach. Again, there are close relationships between structured sparsity methods for optimization-based deep learning and variational Bayesian deep learning. There is also specific prior work on sparsity inducing priors for variational Bayesian methods. Previous work in this area includes the use of horseshoe priors [Carvalho et al., 2009] for approximate Bayesian inference [Ghosh and Doshi-Velez, 2017, Louizos et al., 2017].

Finally, we note that there is a significant gap between the practical implementation of models that result from structured sparsity methods and unstructured sparsity methods. This is due to the fact that structured sparsity methods learn more compact dense models that do not need sparse matrix support to realize storage and computational savings. A key metric for evaluating the computational requirements during inference is the number of floating-point operations (FLOPS). Structured pruning methods can in practice lead to lower FLOPS even with lower compression ratio when compared to unstructured pruning methods. However, structured sparsity methods have been observed to require more non-zero weights than weight-sparse models to obtain similar levels of predictive performance [Blalock et al., 2020]. On the other hand, the run time of linear algebra operations for sparse matrices on real hardware can have significantly more overhead than when operating over dense matrices. The best approach to use is likely to be highly dependent on the hardware and software support available on a particular deployment platform, and has not been studied sufficiently.

2.3 Model Distillation Approaches to Improving the Scalability of Bayesian Deep Learning

In this section, we describe posterior distillation methods, which provide an alternative to pruning methods that can also both decrease the storage cost and the computational cost of applying Bayesian deep learning methods. A prominent method in this area includes Bayesian Dark Knowledge (BDK) [Balan et al., 2015]. These methods aim to compress the computation of expectations under the model posterior into neural networks whose storage and computational complexity can be set to match deployment constraints. As a result, distillation approaches can expose a flexible trade-off between resource use and posterior approximation accuracy.

In the case of BDK, the selected posterior expectation is the posterior predictive distribution. BDK approximates the posterior predictive distribution by learning an auxiliary neural network model to compress the Monte Carlo approximation to the posterior predictive distribution $\mathbb{E}_{p_{MC}(\theta|\mathcal{D},\lambda)}[p(y|\mathbf{x},\theta)]$.

The major advantage of this family of approaches is that they can drastically reduce the deployment time computational complexity of posterior predictive inference relative to using a Monte Carlo average computed using many samples. However, a shortcoming of this family of approaches is that they only capture the target posterior expectations. Thus, they do not have the ability to compute other statistics without being re-trained.

Ensemble distribution distillation (EnD^2) is a closely related approach that aims to distill the collective predictive distribution outputs of the models in an ensemble into a neural network that predicts the parameters of a Dirichlet distribution [Malinin et al., 2020]. The goal is to preserve more information about the distribution of outputs of the ensemble in such a way that multiple statistics of the ensemble's outputs can be efficiently approximated. We note that the EnD^2 approach can be applied to any ensemble of models producing categorical output distributions and can thus be applied to distill the predictive distributions of the elements of a posterior ensemble obtained using MCMC methods as well as those obtained from a variational approximation. We also note that this approach can be extended to approximate the distribution of other posterior quantities by distilling in to approximating models that output other types of distributions.

There is an interesting trade-off between distilling the full parameter posterior distribution into models that predict specific posterior expectations, such as BDK, and approaches that distill aspects of the posterior into distributions. As noted above, expectation distillation is a less general approach and models must be re-trained to extend coverage to additional expectations. On the other hand, EnD^2 gains generality by predicting parametric distributions, from which a wider range of posterior properties can be computed. The drawback of this approach is that it introduces irreducible bias via the selection of a particular approximating family of distributions in a way that is similar to variational inference. As a result, although a wider range of posterior properties can be estimated using EnD^2, their accuracy can be varied and all can be biased if the approximating family is not a good match to the true posterior.

In terms of deployment, one of the distinct advantages of distillation-based approximations over both MCMC and variational methods is that the deployment time computational complexity of approximating posterior expectations can be completely controlled via the selection of the model architecture that the posterior is distilled into. Once this architecture is selected and the distillation process is carried out, the result is a single model (or one model per posterior expectation of interest) that can then be deployed. This can be much simpler than deploying a posterior ensemble in the case of MCMC methods.

However, maximizing the performance of distillation-based methods for a given computational and storage budget requires performing an architecture search to determine the optimal architecture for the approximating model. Since distillation-based learning is itself an optimization problem, one method to perform this search is to start with a large model architecture and apply iterative pruning and fine-tuning as described in the previous section. The use of both weight-level and the structure-level pruning is possible, with the deployment caveats noted in the previous section. The question of how to most efficiently search for an optimal distillation architecture remains an open question.

2.4 Bayesian Decision Theory and Loss-Calibrated Inference

In several practical applications, there's a cost or utility (often asymmetric) associated with the decisions we make based on our model predictions. Bayesian decision theory provides a framework for decision-making under uncertainty [Berger, 1985]. Under the framework, we elicit a utility function $\mathbf{u}(h, y)$, where h denotes a decision within a set of possible actions $\mathcal{A}$ and y denotes model predictions. Next, given a data point $\mathbf{x}_*$, we evaluate the expected utility (also known as the *conditional gain*), $\mathcal{G}(h \mid \mathbf{x}_*)$, for all $h \in \mathcal{A}$ using the utility function $u(.)$ and the posterior predictive distribution $p(y_*|\mathbf{x}_*, \theta, \mathcal{D})$,

$$\mathcal{G}(h \mid \mathbf{x}_*) = \int_{y_*} u(h, y_*) p(y_* \mid \mathbf{x}_*, \mathcal{D}, \theta^0) d\mathbf{y}_* \tag{2.8}$$

Finally, we select the optimal decision c_*, such that it maximizes the conditional gain, $c_* = \arg\max_{h \in \mathcal{A}} \mathcal{G}(h \mid \mathbf{x}_*)$. However, an important assumption in these frameworks is that we have access to the *true* posterior predictive distribution. As noted earlier, the true posterior predictive distribution is intractable for Bayesian neural networks. Rather, in practice, we only have access to an approximation $\tilde{p}(y_* \mid \mathbf{x}_*, \mathcal{D}, \theta^0)$. Using this approximation as a drop-in replacement to $p(y_* \mid \mathbf{x}_*, \mathcal{D}, \theta^0)$ in Eq. (2.8) no longer guarantees optimality of decisions, c_*.

This observation has inspired research in loss-calibrated inference. Lacoste-Julien et al. [2011] presents a variational approach for Gaussian process classification that derives from lower-bounding the log-conditional gain. To train the variational distribution, it presents an EM algorithm with closed form updates, which alternates between sampling from the variational posterior and making optimal decisions under the variational posterior. Cobb et al. [2018a] extends the work done by Lacoste-Julien et al. [2011] to Bayesian neural networks, and derives an objective that is a cost-penalized version of the standard evidence lower-bound (ELBO). Both Lacoste-Julien et al. [2011] and Cobb et al. [2018a] deal with only discrete values for the decisions h, Kuśmierczyk et al. [2019] generalizes these methods to continuous decisions. Beyond variational approximations, Abbasnejad et al. [2015] present an importance sampling-based approach that encourages high utility decisions.

CHAPTER 3

URSABENCH: THE UNCERTAINTY, ROBUSTNESS, SCALABILITY, AND ACCURACY BENCHMARK

As we noted in the previous chapter, the recent advances in Bayesian deep learning have provided a variety of methods to generate posterior representations for supervised deep learning problem, with different trade-offs. While highly promising, much of this prior research has lacked a focus on comprehensive evaluation including the simultaneous assessment of multiple aspects of performance (e.g., in-distribution accuracy, out-of-distribution detection, adversarial robustness, uncertainty calibration, storage costs, computational costs, etc.). There has also been a lack of systematic comparisons across different methods, including accounting for optimizing hyperparameters of inference methods. As a result, there is a significant information gap around which methods hold the most promise in different regions of the multi-faceted trade-off space for different down-stream tasks.

This chapter advances the area of approximate Bayesian inference for deep learning models by addressing the question of how to fairly and efficiently perform comprehensive evaluations of methods. To this end, we present *URSABench* (the Uncertainty, Robustness, Scalability, and Accuracy Benchmark), an open-source suite of models, inference methods, tasks and benchmarking tools. URSABench supports comprehensive assessment of Bayesian deep learning models and approximate Bayesian inference methods, with a focus on classification tasks performed both on server and edge GPUs.

The organization of the chapter is as follows. In Section 3.1 we discuss principles for evaluating Bayesian deep learning models and algorithms including predictive performance, calibration, robustness, and scalability. In Section 3.2 we present the URSABench benchmarking system components that encapsulate the evaluation principles described in Section 3.1. In Section 3.3 we present and discuss benchmarking results obtained using the system, including results on edge GPU hardware.

Bibliographical note: This chapter is adapted from Vadera et al. [2022].

3.1 Evaluation Principles

While advances in supervised deep learning methods have focused almost exclusively on accuracy over the last decade [Krizhevsky et al., 2012], there are multiple additional properties of models and inference algorithms that are of great interest, particularly for Bayesian deep learning models and methods that aim to better represent uncertainty. We group these properties into several categories including predictive performance, calibration, robustness, and computational efficiency. We discuss each of these categories in this section, as well as tasks, methods, and metrics to evaluate them.

3.1.1 Predictive Performance

Predictive performance is by far the most widely considered property of supervised machine learning models. A fundamental question when evaluating probabilistic supervised models is how is the posterior predictive uncertainty evaluated?

In the classification setting, accuracy is the coarsest measure of the predictive performance of a probabilistic model. It simply assesses the number of examples for which the true class has the highest predictive probability. Measures like precision, recall, and the F1 score are all similar to accuracy in that they are based on hard predictions. While these measures have strong interpretability properties, they are not particularly sensitive to posterior uncertainty. In particular, these measures are not affected by the level of uncertainty, so long as the most likely class is the correct prediction. These metrics thus do not differentiate between mild confidence in incorrect predictions and egregiously high confidence in incorrect predictions. In the regression setting, classical evaluation metrics including mean squared error and mean absolute error have a similar issue. While they are sensitive to the predictive mean, they also ignore predictive uncertainty.

In the classification setting, predictive log likelihood can provide a more refined notion of probabilistic predictive performance by assessing the log probability of the true class. This measure is strongly influenced by confident incorrect predictions. In the regression case, the use of predictive log likelihoods for evaluation can also provide more information about the posterior uncertainty of the predictions. However, one of the main drawbacks of predictive log likelihood is that absolute log likelihood values are not as easily interpretable as absolute accuracy values. Predictive likelihood is often better suited for use in relative comparisons between models or inference methods as a result.

Lastly, in general, predictive performance must be assessed on a test dataset $\mathcal{D}_{te}$. The standard assumption is that this test set is sampled from the same distribution as the training dataset $\mathcal{D}_{te}$. We will refer to this setting as *in-domain prediction.* Recent work in machine learning has put an increasing focus on robustness to encountering out of domain examples at deployment time. We will return to this issue in Section 3.1.3.

3.1.2 Calibration

In the binary classification domain, interpreting probability as relative frequency leads to the idea that the fraction of instances that should be classified as positive among a set of instances predicted to be positive with probability p should be exactly p. A model that satisfies this property for all values of p is said to be *perfectly calibrated.* While early work on neural network models indicated they had better calibration properties than some other types of models [Niculescu-Mizil and Caruana, 2005], more

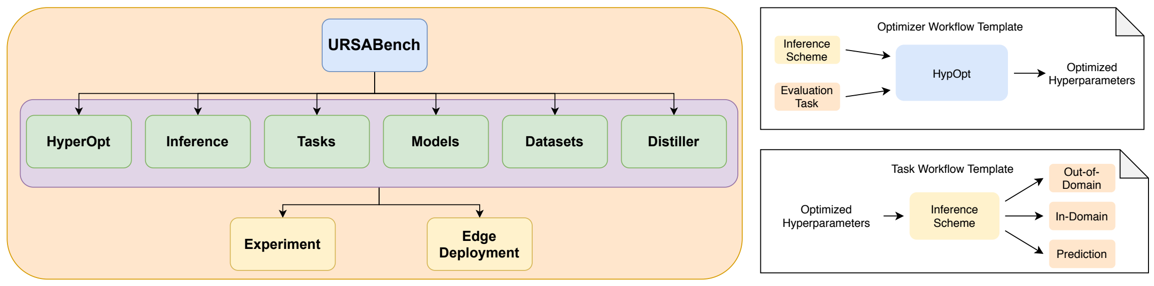


Figure 3.1. (Left) Individual components of, URSABench shown at the middle level. Based on a configuration of a subset of these components, users can run an end-to-end experiment or deploy the sampled models to an edge environment. (Right, top) A sample workflow template for HyperOpt. Users select an inference scheme and an evaluation task, and select the hyperparameters which perform the best for the chosen evaluation scheme. (Right, bottom) A sample workflow template for Inference. Users select hyperparameters (ideally through some form of hyperparameter optimization), and an inference scheme to obtain a Bayesian ensemble. Finally, this Bayesian ensemble is evaluated on different tasks, as shown on the right.

recent evaluations of deep learning models have shown that their calibration properties can be quite poor [Guo et al., 2017].

In the case of approximate Bayesian supervised classification, calibration is estimated by computing the predictive probability $p_i = p(Y_i = 1|\mathbf{x}, \mathcal{D}, \theta^0)$ for each test instance i, binning the predictive probabilities, and computing the accuracy within each bin b. Let N_b be the number of instances in bin b, A_b be the accuracy of instances in bin b, and C_b be the average predictive probability (or confidence) of instances in bin b. Plotting the bin accuracies A_b as a bar chart results in a visualization referred to as a *reliability plot*. We can also compute the calibration error for bin b as $\kappa_b = |A_b - C_b|$. The expected calibration error (ECE) is then defined to be $ECE = \frac{1}{N}\sum_b N_b\kappa_b$. The maximum calibration error is given by $MCE = \max_b \kappa_b$ [Guo et al., 2017].

In the case of multi-class classification, we can obtain a generalization of calibration using a one-vs-all formulation. We let $p_{ci} = p(Y_i = c|\mathbf{x}, \mathcal{D}, \theta^0)$ and bin the p_{ci} values separately for each class c. Let N_{cb} be the number of instances in bin b, A_{cb} be the accuracy of instances in bin b, and C_{cb} be the average predictive probability (or confidence) of instances in bin b, all when class c is considered to be the target class. We then define $\kappa_{cb} = |A_{cb} - C_{cb}|$ and $ECE = \frac{1}{NC} \sum_c \sum_b N_{cb}\kappa_{cb}$.

It is important to note that calibration and accuracy are distinct concepts. A model with low accuracy could be perfectly calibrated, while a model with high accuracy could be poorly calibrated. Calibration and predictive performance are thus largely orthogonal properties of a model.

The Brier score provides a measure related to the multi-class generalization of expected calibration error [Brier, 1950]. Again, letting $p_{ci} = p(Y_i = c|\mathbf{x}, \mathcal{D}, \theta^0)$, the multi-class Brier score is given by $BS = \frac{1}{N} \sum_c \sum_i (p_{ci} - [y_i = c])^2$ where $[y_i = c]$ is 1 if $y_i = c$ and is 0 otherwise. The Brier score can be interpreted as mixing together aspects of calibration and predictive performance.

Like prediction performance measures, calibration is assessed on an in-domain test dataset $\mathcal{D}_{te}$ that is assumed to be sampled from the same distribution as the training dataset $\mathcal{D}_{tr}$.

3.1.3 Robustness

A key aspect of robustness of methods is their ability to identify when input examples are not drawn from the training distribution [Ovadia et al., 2019]. Such an evaluation leverages a test set that is explicitly out-of-distribution (OOD). A variety of methods have been proposed to leverage the output of a probabilistic model for the purpose of detecting when examples are OOD. Examples include thresholds on the entropy of the posterior predictive distribution (also called the total uncertainty). For approaches that provide access to the posterior distribution over model parameters, methods have been proposed that instead threshold the knowledge uncertainty. Performance is typically measured via OOD prediction as a binary classification task.

Another key property of models and inference methods is their robustness to adversarial manipulations [Goodfellow et al., 2015] and out of distribution examples [Ovadia et al., 2019]. In the case of adversarial examples, there are multiple possible frameworks for generating examples and the robustness property of interest is the ability of methods to resist adversarial perturbations as measured by the success rate of adversarial attacks as a function of bounds on the adversarial perturbation magnitude. Adversarial example generation methods include the Fast Gradient Sign Method (FGSM) [Goodfellow et al., 2015], Projected Gradient Descent (PGD) [Madry et al., 2018] and the Carlini-Wagner attack [Carlini and Wagner, 2017], and one-pixel attack [Su et al., 2019].

We note that in the Bayesian supervised learning context, these attack methods require access to the posterior predictive distribution function and its gradients. It is possible to efficiently attack sample-based representations of Bayesian posterior predictive distributions by iteratively constructing the adversarial perturbations while running an MCMC sampler.

3.1.4 Scalability

We consider two primary aspects to computational scalability: run time and storage cost. As noted in Chapter 2, different approximate Bayesian inference methods can have widely different properties in terms of both aspects of computational scalability. Of primary interest in this work are how the predictive performance, calibration and robustness properties of methods trade off against their computational scalability properties. In particular, which approach is "best" depends on what aspects of performance are important for a given task.

For example, the method that gives the best absolute predictive performance on a given task may be acceptable even if it has high run time and storage requirements if the deployment context is a server and there are no real-time constraints on making predictions. However, if there are real-time constraints, a method that is faster may be needed. If the deployment context is an embedded system, methods that are both faster and require reduced storage may be needed.

When benchmarking methods, the storage cost will be estimated via the number of parameters. The run-time of methods can be assessed in different ways, including actual computation time as well as a number of more portable statistics such as the number of floating-point operations (flops) or the number of multiply-accumulate operations (MACs). In this work we benchmark training run time on server hardware and prediction/inference run time on both server and edge hardware.

3.2 The URSABench System

In this section, we describe the URSABench system components, which include multiple datasets, models, inference methods, and tasks that implement the evaluation principles described in the previous section. URSABench is implemented using the PyTorch deep learning library [Paszke et al., 2019]. The current system includes benchmarking components for small-scale, medium-scale, and large-scale models and tasks, as well as evaluation of scalability on server and edge GPU-accelerated platforms. The architecture of URSABench is illustrated in Figure 3.1. There are six main modules including **HyperOpt**, **Inference**, **Tasks**, **Models**, **Datasets** and **Distiller**. We describe each of them below.

3.2.1 The HyperOpt Module

For comparisons between methods to be meaningful, hyperparameters of the inference methods and models need to be set for each combination of inference method, model, dataset, and task. The URSABench HyperOpt module provides several hyperparameter optimization approaches including Bayesian optimization, random search, and grid search. The results in the next section use Bayesian optimization as the hyperparameter optimization method.

3.2.2 The Inference Module

The Inference module includes implementations of multiple recent inference methods for Bayesian deep learning with a standard interface. Methods in this module allow users to either sample iteratively, or collect all parameter posterior samples at once. This allows greater flexibility in interfacing with downstream tasks or distillation while working under memory constraints. Implemented inference methods include HMC[1], SGLD [Welling and Teh, 2011], SGHMC [Chen et al., 2014b], cSGLD, cSGHMC [Zhang et al., 2020], SWAG [Maddox et al., 2019] and PCA-based subspace inference + elliptical slice sampling (PCA+ ESS (SI)) [Izmailov et al., 2019]. As baselines, we also provide MC dropout [Gal and Ghahramani, 2016] and an SGD-based point estimation.

3.2.3 The Distiller Module

This module provides methods to distill a posterior ensemble into a student model. Knowledge distillation is useful to reduce the deployment time storage and computation requirements. An implementation of Bayesian Dark Knowledge (BDK) [Balan et al., 2015] based distillation, Generalized Bayesian Posterior Expectation Distillation (GPED) [Vadera et al., 2020b] is included.

3.2.4 The Models and Datasets Modules

The small-scale benchmark uses a basic, fully connected MLP with two hidden layers containing 200 units each as the benchmark model, with MNIST providing the benchmark in-domain dataset [LeCun, 1998]. At the medium-scale, we use ResNet50 [He et al., 2016b] and WideResNet as the benchmark models [Zagoruyko and Komodakis, 2016], with CIFAR10 and CIFAR100 as the benchmark in-domain datasets [Krizhevsky et al., 2009]. At the large-scale, we use ResNet50 with ImageNet [Deng et al., 2009] as the benchmark in-domain dataset.

[1]HMC is only implemented for tasks where the model and full dataset can fit on the GPU. We use the `hamiltorch` Python package [Cobb et al., 2019].

3.2.5 The Tasks Module

The Tasks module includes a set of inference and prediction tasks using different datasets. The tasks are designed to support assessment of multiple proprieties of models and algorithms, including most of those identified in the discussion of evaluation principles. Specifically, accuracy is assessed using in-domain test sets. To assess uncertainty quantification, we compute negative log likelihood, Brier score, and performance on a misclassification detection task, all using the in-domain test sets. We also consider a decision-making task that focuses on assessing the quality of the tail of the predictive distribution using imbalanced datasets and costs that strongly penalize errors on the rare classes [Cobb et al., 2018b].

We assess robustness using an out-of-distribution (OOD) classification task [Ovadia et al., 2019, Vadera et al., 2020b] leveraging knowledge uncertainty (see Chapter 2, Section 2.1 for a review of uncertainty decomposition). The small-scale benchmark uses FashionMNIST [Xiao et al., 2017] and KMNIST [Clanuwat et al., 2018b] as OOD test sets, while the medium-scale benchmark uses SVHN [Netzer et al., 2011b] and STL10 [Coates et al., 2011] as OOD test sets. For the large-scale benchmark, we use ImageNet-Sketch [Wang et al., 2019] as OOD test set. Performance on OOD tasks is assessed using AUROC. Finally, the benchmark focuses on computation time as the measure of computational scalability, measured in seconds/sample.

3.2.6 The Run-time Latency Module

We implement a profiling pipeline to evaluate and compare model inference latency. The models optimized by HyperOpt are compiled using TensorRT [NVIDIA Corporation] before deployment. TensorRT is an acceleration library from Nvidia that improves the inference efficiency of already-trained models on Nvidia platforms. At compilation, TensorRT applies several hardware-dependent optimizations to the input model such as quantization, layer fusion, kernel auto-tuning, etc. The compiled model is saved as a TensorRT engine and executed by the TensorRT run-time during the execution phase. This module supports the deployment of Bayesian ensembles and distilled models onto edge hardware for run-time latency benchmarking.

3.2.7 Experiment Workflows and Composite Scores

Along with the above components, we also provide workflows for executing hyperparameter optimization, inference and evaluation for each of the model-method-dataset-task combination that form the core of the benchmark evaluation process. This includes workflow components for compiling models for edge deployment using TensorRT.

We divide our benchmark into three categories: small-scale, medium-scale, and large-scale. The small-scale benchmark consists of MNIST as its in-domain dataset using the MLP models. The medium-scale benchmark consists of all combinations CIFAR datasets with the WideResNet28x10 and ResNet50 models. Finally, the large-scale benchmark consists of ImageNet dataset with the ResNet50 model.

Lastly, the simultaneous assessment of multiple aspects of performance for multiple model, method, dataset and task combinations yields many individual results for each inference method. An important design choice in URSABench is thus to summarize performance in terms of key selected metrics along with composite scores that combine related individual metrics. For accuracy, we include an average over all benchmark models and all in-domain test sets. For robustness, we use an average over all models and OOD datasets. For uncertainty quantification, we separately compute an average over models and datasets in terms of negative log likelihood (NLL) and misclassification task performance.

3.2.8 Discussion: Extensibility and Reproducibility

The URSABench system is designed to be extensible and reproducible. Researchers who are interested in benchmarking the performance of a new model can add it to the Models module and update the benchmarking workflow template to evaluate the model. Similarly, a researcher interested in benchmarking a new approximate inference algorithm can add it to the Inference module, update the benchmarking workflow template to include the new algorithm, and evaluate its performance. The benchmark predictive performance, robustness and uncertainty evaluations are all fully reproducible based on the existing experimental workflow implementations.

The one aspect of the system that is not fully reproducible is the scalability assessment, as it relies on profiling methods on real hardware. In the prediction run-time results reported in the next section, we use the Jetson TX2 platform. These results are reproducible on that hardware, which is available at relatively low cost. The training scalability results use server GPUs where there is much more variability in available hardware. In this work, we show absolute time for training scalability, but a metric with more potential stability across hardware platforms is the ratio of the absolute training time score for a given method to that of the base SGD method used on the same platform. We leave an exploration of the standardization of training scalability across hardware platforms to future work.

3.3 URSABench Benchmark Results

In this section, we present the implementation details, and report benchmark results obtained using the URSABench system. We divide our benchmark results into the three categories mentioned earlier: small-scale, medium-scale and large-scale. To get an overall assessment of different approximate inference methods, we present an aggregated set of metrics that focus on the accuracy, negative log-likelihood (NLL), robustness, uncertainty and training scalability. The robustness relies on averaging both the total uncertainty AUROC and the model uncertainty AUROC over the OOD data sets. We then further average the mean total uncertainty and mean model uncertainty scores. The uncertainty composite score is built from the average misclassification AUROCs (e.g. the first three columns of Tables 3.4, 3.8, 3.11, 3.14, 3.17). For the medium-scale experiment, the uncertainty score is then averaged across CIFAR10 and CIFAR100 as well as ResNet50 and WideResNet28x10. Finally, the training scalability denotes the wall-clock time (in seconds) required to generate one sample from the parameter posterior.

3.3.1 Implementation Details

In this section, we describe the implementation details for the different inference methods used in our benchmark. It must be noted that for all inference methods using ResNet50 and WideResNet28x10 models, we use a pretrained SGD solution to warm-start our samplers. This is a standard pretraining procedure followed to make the methods more competitive Maddox et al. [2019]. Further, the ensemble size is set to 100 for MNIST dataset, 50 for CIFAR datasets, and 6 for ImageNet dataset. When applicable, we always collect a sample at the end of an epoch. The difference in ensemble size is due to the large amounts of computational requirements for training ResNet50 and WideResNet28x10 on CIFAR datasets, as well training ResNet50 on ImageNet. While tuning hyperparameters for MNIST, we apply Bayesian optimization with a limit of 200 evaluations for each approach Balandat et al. [2019]. On the other hand, for CIFAR datasets, we refer to existing literature and use the same hyperparameters if directly applicable, or search around the hyperparameters obtained for similar models and datasets.

SGLD: For CIFAR datasets, we use a burn-in of 100 epochs and initial learning rates of 0.1 for WideResNet28x10 model and 0.05 for ResNet50. The prior std. dev. is set to 0.5 for both the cases. We decay the learning rate using cosine annealing schedule to its half value by the end of sampling. For ImageNet datasets, we use a burn-in of 5 epochs, initial learning rates of 0.05, and set the prior std. dev. to 0.1. For MNIST, the optimal hyperparameter values obtained are: initial learning rate of 0.099, prior std. dev. of 0.16 and 50 burn in epochs.

SGHMC: We use the same hyperparameters and learning rate schedule as described for SGLD for the CIFAR and ImageNet datasets. Additionally, we set the friction term to 0.5 Chen et al. [2014b]. This is equivalent to the α term shown in Zhang et al. [2020]. For MNIST, the optimal hyperparameter values obtained are: initial learning rate of 0.03, prior std. dev. of 0.14, 50 burn in epochs, and friction term set to 0.1.

cSGHMC: We use the same hyperparameters given in Zhang et al. [2020] for CIFAR datasets. For ImageNet, we use the initial learning rate for a cycle to 0.01, prior std. dev. of 0.1, cycle length of 7 epochs, of which 4 epochs are used for SGD-exploration phase, 1 epoch for burn-in, and collect 2 samples from each of the last two epochs of the cycle. For ImageNet, we run a total of 3 cycles. For MNIST, the optimal hyperparameter values obtained are: initial learning rate of 0.06, prior std. dev. of 0.33, cycle length of 22 epochs, of which 17 epochs are used for SGD-exploration phase, and samples are collected from the last 4 epochs, and friction term set to 0.21.

cSGLD: We use the same hyperparameters given in Zhang et al. [2020] for CIFAR datasets. For ImageNet, we use the same hyperparameters described for cSGHMC. For MNIST, the optimal hyperparameter values obtained are: initial learning rate of 0.06, prior std. dev. of 0.33, cycle length of 22 epochs, of which 17 epochs are used for SGD-exploration phase, and samples are collected from the last 4 epochs, and friction term set to 0.21.

SWAG: We use the same hyperparameters given in Izmailov et al. [2019] for CIFAR models, except that we set the weight decay for ResNet models to 10^{-4} and borrow its remaining hyperparameters from WideResNet28x10. This means that we utilize last 20 SGD iterates to find parameters for the Gaussian approximation to the mode. For ImageNet, we set the exploration learning rate to 0.005, and start collecting samples for constructing the Gaussian approximation after 10 epochs. We collect 20 parameter samples for generating the Gaussian approximation. Furthermore, the variant of SWAG used in our benchmark is SWAG-diagonal.

PCA + ESS (SI): We use the same hyperparameters given in Izmailov et al. [2019] for CIFAR models, except that we set the weight decay for ResNet models to 10^{-4} and borrow its remaining hyperparameters from WideResNet28x10. We construct a subspace of rank 20 for all models and datasets. For ImageNet, we set the exploration learning rate to 0.005, and start collecting samples for constructing the subspace after 10 epochs. We collect 20 parameter samples for generating the subspace of rank 20. For MNIST, we start with an initial learning rate of 0.04 and decay it 0.002 over 50 epochs. Further, we run SGD at the same learning rate for another 50 epochs and collect the iterates from each of the final 20 epochs to construct our PCA subspace. The momentum for our SGD optimizer is set to 0.54 through the entire run. For all the dataset and model combinations, we use elliptical slice sampling [Murray et al., 2010] on the low rank PCA subspace with a prior of 2. and a temperature of 5000.

MC Dropout: For all the models on CIFAR & MNIST datasets, we use a dropout of 0.2 before the final linear layer. For ImageNet, we use a dropout rate of 0.05 just before the final layer. We always fine-tune after warm-starting with SGD point-estimated model.

3.3.2 Small-Scale Benchmark Results

The composite scores for the small-scale benchmark are displayed in Table 3.1. The detailed experimental results behind each composite score can be found in Tables 3.2 - 3.4. The small-scale results show how challenging it can be to distinguish between different approximate inference schemes using relatively simple models and datasets. SGD and SGHMC are both marginally ahead in accuracy and NLL; HMC appears to show the most robust performance in OOD, and SGHMC does best for the uncertainty metric. However, the minor relative difference between all the performance metrics points to focusing on the compute time, which shows HMC to be significantly less time-consuming. This is due to the ability to fit all the data and model parameters on the GPU. Recall that the training-time scalability represents the total no. of seconds required during training. To benchmark this, we use an Nvidia TITAN X GPU, with a batch size of 128, and 4 PyTorch dataloader workers.

Table 3.1. URSABench **small-scale** benchmark performance. (Results presented as mean ± std. dev. across 5 trials.)

Inference	Accuracy ↑	NLL ↓	Robustness ↑	Uncertainty ↑	Training Time ↓
HMC	0.9819 ± 0.0010	0.0593 ± 0.0016	0.9570 ± 0.0075	0.9734 ± 0.0012	178 ± 1.5
SGLD	0.9839 ± 0.0004	0.0492 ± 0.0022	0.9065 ± 0.0377	0.9679 ± 0.0233	765 ± 4.2
SGHMC	0.9862 ± 0.0003	0.0446 ± 0.0003	0.9426 ± 0.0048	0.9807 ± 0.0003	768 ± 2.3
cSGLD	0.9857 ± 0.0003	0.0476 ± 0.0011	0.9521 ± 0.0022	0.9795 ± 0.0007	1063 ± 4.2
cSGHMC	0.9836 ± 0.0009	0.0533 ± 0.0016	0.9276 ± 0.0094	0.9759 ± 0.0015	1111 ± 3.3
PCA + ESS (SI)	0.9840 ± 0.0007	0.0520 ± 0.0016	0.9360 ± 0.0038	0.9695 ± 0.0012	1146 ± 10.2
MC dropout	0.9858 ± 0.0007	0.0501 ± 0.0031	0.9429 ± 0.0059	0.9769 ± 0.0019	377 ± 2.4
SGD	0.9860 ± 0.0002	0.0452 ± 0.0012	-	-	193 ± 0.9

Table 3.2. Comparison of predictive performance and decision making cost while using an MLP $[784, 200, 200, 10]$ on MNIST. Results presented as mean ± std. dev. across 5 trials.

Inference	Accuracy ↑	NLL ↓	BS ↓	ECE ↓	Decision Cost ↓	Training Time ↓
HMC	98.19 ± 0.10%	0.0593 ± 0.0016	0.0280 ± 0.0008	0.0079 ± 0.0008	7101 ± 346	178 ± 1.5
SGLD	98.39 ± 0.04%	0.0492 ± 0.0022	0.0236 ± 0.0005	0.0041 ± 0.0024	5410 ± 778	765 ± 4.2
SGHMC	98.62 ± 0.03%	0.0446 ± 0.0003	0.0210 ± 0.0002	0.0073 ± 0.0004	5408 ± 240	768 ± 2.3
cSGLD	98.57 ± 0.03%	0.0476 ± 0.0011	0.0223 ± 0.0003	0.0056 ± 0.0003	6526 ± 2241	1063 ± 4.2
cSGHMC	98.36 ± 0.09%	0.0533 ± 0.0016	0.0256 ± 0.0010	0.0033 ± 0.0003	4824 ± 1855	1111 ± 3.3
PCA + ESS (SI)	98.40 ± 0.07%	0.0520 ± 0.0016	0.0251 ± 0.0007	0.0036 ± 0.0005	3809 ± 1150	1146 ± 10.2
MC dropout	98.58 ± 0.07%	0.0501 ± 0.0031	0.0218 ± 0.0008	0.0042 ± 0.0006	15236 ± 1184	377 ± 2.4
SGD	98.60 ± 0.02%	0.0452 ± 0.0012	0.0213 ± 0.0003	0.0032 ± 0.0005	8972 ± 720	193 ± 0.9

3.3.3 Medium-Scale Benchmark Results

The medium-scale results are displayed in Table 3.5. The detailed experimental results behind each composite score can again be found in Tables 3.6 - 3.17.

We note that we provide two sets of results for cSGHMC and cSGLD, with different ensemble sizes, denoted by the number in parentheses. For getting 50 parameter samples from the posterior using either cSGLD or cSGHMC, we run many cycles,

Table 3.3. Comparison of OOD detection performance while using an MLP $[784, 200, 200, 10]$ on MNIST. Results presented as mean ± std. dev. across 5 trials.

Inference	OOD Dataset	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑
HMC	Fashion MNIST	0.966 ± 0.013	0.946 ± 0.017
	KMNIST	0.968 ± 0.013	0.948 ± 0.017
SGLD	Fashion MNIST	0.867 ± 0.110	0.944 ± 0.005
	KMNIST	0.871 ± 0.103	0.944 ± 0.005
SGHMC	Fashion MNIST	0.933 ± 0.009	0.953 ± 0.010
	KMNIST	0.932 ± 0.009	0.952 ± 0.010
cSGLD	Fashion MNIST	0.954 ± 0.004	0.950 ± 0.005
	KMNIST	0.954 ± 0.004	0.950 ± 0.005
cSGHMC	Fashion MNIST	0.923 ± 0.021	0.931 ± 0.017
	KMNIST	0.923 ± 0.020	0.933 ± 0.017
PCA + ESS (SI)	Fashion MNIST	0.933 ± 0.006	0.938 ± 0.009
	KMNIST	0.934 ± 0.006	0.940 ± 0.009
MC dropout	Fashion MNIST	0.942 ± 0.013	0.943 ± 0.010
	KMNIST	0.943 ± 0.013	0.944 ± 0.010
SGD	Fashion MNIST	-	0.945 ± 0.010
	KMNIST	-	0.943 ± 0.010

Table 3.4. Comparison of misclassification detection while using an MLP $[784, 200, 200, 10]$ on MNIST.

Inference	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑	AUROC- Model Confidence ↑	AUCPR- Model Uncertainty ↑	AUCPR - Total Uncertainty ↑	AUCPR- Model Confidence ↑
HMC	0.9706	0.9734	0.9743	0.3429	0.3888	0.4145
SGLD	0.9739	0.9800	0.9800	0.3530	0.4502	0.4632
SGHMC	0.9786	0.9815	0.9823	0.3546	0.3929	0.4131
cSGLD	0.9769	0.9801	0.9798	0.3695	0.4477	0.4478
cSGHMC	0.9730	0.9786	0.9786	0.3260	0.4255	0.4404
PCA + ESS (SI)	0.9539	0.9774	0.9772	0.2059	0.4298	0.4248
MC dropout	0.9754	0.9763	0.9760	0.4085	0.4300	0.4199
SGD	-	0.9795	0.9794	-	0.4273	0.4389

which results in very high training time as compared to SGLD and SGHMC. Thus, for a more fair comparison with SGLD and SGHMC, we limit the training time of cSGLD and cSGHMC to be similar to their non-cyclic versions by reducing the number of cycles and parameter samples. In the setting with the reduced number of cycles, we extract nine parameter samples, as compared to 50 for the full number of cycles. Overall, the medium-scale experiments indicate a slight improvement in predictive performance and decision-making tasks from both cSGHMC (50) and cSGLD (50) followed by PCA + ESS (SI) & cSGLD. Importantly, the cSGLD & cSGHMC results with a lower ensemble size do better overall compared to cSGLD and cSGHMC with 50 parameter samples. Thus, once again, a user may prefer using MC dropout, SGLD/SGHMC or their cyclic versions with fewer parameter samples extracted, as they provide respectable performance in significantly less time. This is due to the large proportion of time that the cyclic schemes spend exploring without sampling. Furthermore, if the goal is to compute uncertainty metrics and ultimately use them for misclassification detection or OOD detection, then SGHMC/SGLD provide better performance in most cases. Another important result that can be seen from the Tables 3.6, 3.9, 3.12 and 3.15 is the utility of the decision-making task in highlighting the top performing approximate inference schemes for each model and dataset, via its correlation with low NLL and high accuracy.

For benchmarking the training-time scalability, we use an Nvidia TITAN X GPU, with a batch size of 128, and 4 PyTorch dataloader workers. Note that we provide an estimated training time for the methods which involve stochastic gradients. More specifically, we compute the per-epoch average training time for methods involving stochastic gradients, and then extrapolate it by multiplying it with the total number of epochs involving the same stochastic gradient based computation in that method. For example, while computing training time for SGLD and cSGLD, we first compute the average training time for SGLD for the same model-dataset combination, and then extrapolate it to the average training time shown in this chapter by multiplying it by the total number of epochs run for cSGLD. We use the same approach for SGHMC and cSGHMC, as well as for SGD, MC Dropout, SWAG and PCA+SSI (ESS).

Table 3.5. URSABench **medium-scale** benchmark performance.

Inference	Accuracy ↑	NLL ↓	Robustness ↑	Uncertainty ↑	Training Time ↓
SGLD	0.863	0.620	0.781	0.908	51117
SGHMC	0.866	0.599	0.777	0.905	51466
cSGLD(50)	0.881	0.453	0.819	0.919	230155
cSGHMC(50)	0.880	0.446	0.812	0.917	231441
cSGLD(9)	0.876	0.484	0.802	0.911	51117
cSGHMC(9)	0.873	0.506	0.802	0.905	51466
SWAG	0.824	0.735	0.759	0.885	75544
PCA + ESS (SI)	0.869	0.482	0.804	0.901	98696
MC dropout	0.872	0.554	0.775	0.914	25733
SGD	0.861	0.625	-	-	12866

Table 3.6. Comparison of predictive performance and decision making cost while using ResNet50 on CIFAR10.

Inference	Accuracy ↑	NLL ↓	BS ↓	ECE ↓	Decision Cost ↓	Training Time ↓
SGLD	0.943	0.275	0.094	0.041	176.000	31193
SGHMC	0.949	0.256	0.086	0.035	154.000	31538
cSGLD(50)	0.961	0.137	0.060	0.011	118.500	139336
cSGHMC(50)	0.962	0.124	0.056	0.009	125.000	141291
cSGLD(9)	0.952	0.198	0.075	0.023	147.200	31193
cSGHMC(9)	0.956	0.178	0.070	0.019	135.100	31538
SWAG	0.931	0.311	0.114	0.047	200.900	44344
PCA + ESS (SI)	0.949	0.174	0.080	0.027	166.600	55564
MC dropout	0.948	0.208	0.083	0.032	159.500	15768
SGD	0.943	0.274	0.095	0.040	171.700	7884

3.3.4 Large-Scale Benchmark Results

The results for the large-scale benchmark are presented in Table 3.18. The detailed experimental results behind each composite score can be found in Tables 3.19 - 3.21. Also, similar to the medium-scale benchmarks, we provide two sets of results for cSGLD and cSGHMC, with different number of parameter samples extracted, where the results corresponding to fewer parameter samples have very similar training time

Table 3.7. Comparison of OOD detection performance while using ResNet50 on CIFAR10.

Inference	OOD Dataset	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑
SGLD	STL10	0.630	0.679
	SVHN	0.863	0.876
SGHMC	STL10	0.639	0.680
	SVHN	0.855	0.849
cSGLD(50)	STL10	0.678	0.700
	SVHN	0.952	0.953
cSGHMC(50)	STL10	0.668	0.692
	SVHN	0.960	0.960
cSGLD(9)	STL10	0.666	0.687
	SVHN	0.902	0.908
cSGHMC(9)	STL10	0.667	0.689
	SVHN	0.936	0.939
SWAG	STL10	0.618	0.671
	SVHN	0.878	0.908
PCA + ESS (SI)	STL10	0.673	0.677
	SVHN	0.949	0.947
MC dropout	STL10	0.665	0.695
	SVHN	0.926	0.938
SGD	STL10	N/A	0.682
	SVHN	N/A	0.892

Table 3.8. Comparison of Misclassification detection while using ResNet50 on CIFAR10.

Inference	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑	AUROC- Model Confidence ↑	AUCPR- Model Uncertainty ↑	AUCPR - Total Uncertainty ↑	AUCPR- Model Confidence ↑
SGLD	0.936	0.934	0.935	0.466	0.483	0.487
SGHMC	0.930	0.929	0.929	0.406	0.411	0.418
cSGLD(50)	0.949	0.950	0.951	0.398	0.403	0.435
cSGHMC(50)	0.948	0.952	0.954	0.394	0.422	0.459
cSGLD(9)	0.939	0.943	0.943	0.378	0.435	0.448
cSGHMC(9)	0.935	0.940	0.940	0.353	0.405	0.420
SWAG	0.890	0.927	0.927	0.418	0.479	0.472
PCA + ESS (SI)	0.932	0.934	0.941	0.391	0.419	0.472
MC dropout	0.946	0.947	0.947	0.455	0.485	0.477
SGD	0.523	0.937	0.936	0.151	0.464	0.456

Table 3.9. Comparison of predictive performance and decision making cost while using ResNet50 on CIFAR100.

Inference	Accuracy ↑	NLL ↓	BS ↓	ECE ↓	Decision Cost ↓	Training Time ↓
SGLD	0.736	1.254	0.400	0.141	299.800	30153
SGHMC	0.738	1.222	0.397	0.137	295.000	30468
cSGLD(50)	0.770	0.902	0.324	0.066	250.100	136891
cSGHMC(50)	0.782	0.838	0.306	0.053	241.100	137106
cSGLD(9)	0.779	0.858	0.316	0.047	242.900	30153
cSGHMC(9)	0.794	0.794	0.295	0.048	223.700	30468
SWAG	0.735	1.221	0.400	0.135	295.200	44343
PCA + ESS (SI)	0.761	0.920	0.335	0.032	261.200	60880
MC dropout	0.786	1.006	0.330	0.115	233.100	15234
SGD	0.732	1.302	0.408	0.148	303.700	7617

Table 3.10. Comparison of OOD detection performance while using ResNet50 on CIFAR100.

Inference	OOD Dataset	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑
SGLD	STL10	0.750	0.772
	SVHN	0.746	0.784
SGHMC	STL10	0.750	0.777
	SVHN	0.754	0.784
cSGLD(50)	STL10	0.786	0.800
	SVHN	0.820	0.834
cSGHMC(50)	STL10	0.793	0.811
	SVHN	0.808	0.827
cSGLD(9)	STL10	0.767	0.804
	SVHN	0.813	0.846
cSGHMC(9)	STL10	0.783	0.814
	SVHN	0.809	0.825
SWAG	STL10	0.748	0.778
	SVHN	0.732	0.771
PCA + ESS (SI)	STL10	0.779	0.797
	SVHN	0.816	0.807
MC dropout	STL10	0.785	0.801
	SVHN	0.755	0.752
SGD	STL10	N/A	0.765
	SVHN	N/A	0.763

Table 3.11. Comparison of Misclassification detection while using ResNet50 on CIFAR100.

Inference	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑	AUROC- Model Confidence ↑	AUCPR- Model Uncertainty ↑	AUCPR - Total Uncertainty ↑	AUCPR- Model Confidence ↑
SGLD	0.866	0.873	0.872	0.635	0.677	0.675
SGHMC	0.864	0.872	0.871	0.622	0.668	0.666
cSGLD(50)	0.875	0.879	0.883	0.633	0.650	0.661
cSGHMC(50)	0.877	0.881	0.885	0.617	0.639	0.651
cSGLD(9)	0.851	0.871	0.876	0.530	0.617	0.636
cSGHMC(9)	0.860	0.876	0.882	0.558	0.612	0.634
SWAG	0.855	0.870	0.869	0.625	0.671	0.667
PCA + ESS (SI)	0.858	0.863	0.877	0.618	0.634	0.667
MC dropout	0.875	0.880	0.877	0.613	0.639	0.624
SGD	N/A	0.873	0.870	N/A	0.687	0.680

Table 3.12. Comparison of predictive performance and decision-making cost while using WideResNet28x10 on CIFAR10.

Inference	Accuracy ↑	NLL ↓	BS ↓	ECE ↓	Decision Cost ↓	Training Time ↓
SGLD	0.963	0.153	0.060	0.023	121.300	72365
SGHMC	0.967	0.140	0.055	0.019	106.500	72740
cSGLD(50)	0.967	0.105	0.049	0.009	107.300	325205
cSGHMC(50)	0.962	0.122	0.057	0.008	125.100	327330
cSGLD(9)	0.961	0.128	0.060	0.008	132.500	72365
cSGHMC(9)	0.951	0.182	0.076	0.019	150.000	72740
SWAG	0.919	0.260	0.121	0.028	270.000	106747
PCA + ESS (SI)	0.951	0.177	0.082	0.054	163.100	141980
MC dropout	0.957	0.158	0.067	0.019	149.000	36370
SGD	0.963	0.138	0.060	0.018	117.100	18185

Table 3.13. Comparison of OOD detection performance while using WideResNet28x10 on CIFAR10.

Inference	OOD Dataset	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑
SGLD	STL10	0.665	0.660
	SVHN	0.936	0.951
SGHMC	STL10	0.665	0.662
	SVHN	0.943	0.954
cSGLD(50)	STL10	0.678	0.679
	SVHN	0.966	0.968
cSGHMC(50)	STL10	0.679	0.690
	SVHN	0.963	0.960
cSGLD(9)	STL10	0.672	0.674
	SVHN	0.944	0.958
cSGHMC(9)	STL10	0.667	0.682
	SVHN	0.924	0.925
SWAG	STL10	0.649	0.667
	SVHN	0.914	0.943
PCA + ESS (SI)	STL10	0.663	0.673
	SVHN	0.897	0.970
MC dropout	STL10	0.672	0.688
	SVHN	0.897	0.922
SGD	STL10	N/A	0.667
	SVHN	N/A	0.963

Table 3.14. Comparison of Misclassification detection while using WideResNet28x10 on CIFAR10.

Inference	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑	AUROC- Model Confidence ↑	AUCPR- Model Uncertainty ↑	AUCPR - Total Uncertainty ↑	AUCPR- Model Confidence ↑
SGLD	0.946	0.946	0.946	0.416	0.445	0.438
SGHMC	0.941	0.940	0.940	0.370	0.401	0.398
cSGLD(50)	0.955	0.957	0.958	0.406	0.416	0.438
cSGHMC(50)	0.946	0.950	0.951	0.393	0.436	0.476
cSGLD(9)	0.941	0.948	0.949	0.343	0.415	0.430
cSGHMC(9)	0.932	0.939	0.939	0.348	0.430	0.455
SWAG	0.900	0.915	0.916	0.375	0.468	0.468
PCA + ESS (SI)	0.918	0.931	0.948	0.337	0.387	0.478
MC dropout	0.946	0.947	0.947	0.432	0.467	0.467
SGD	N/A	0.941	0.942	N/A	0.390	0.387

Table 3.15. Comparison of predictive performance and decision-making cost while using WideResNet28x10 on CIFAR100.

Inference	Accuracy ↑	NLL ↓	BS ↓	ECE ↓	Decision Cost ↓	Training Time ↓
SGLD	0.809	0.796	0.280	0.064	202.300	70760
SGHMC	0.810	0.779	0.277	0.061	203.100	71120
cSGLD(50)	0.827	0.666	0.248	0.030	183.100	319190
cSGHMC(50)	0.813	0.699	0.264	0.024	198.400	320040
cSGLD(9)	0.813	0.754	0.268	0.062	200.200	70760
cSGHMC(9)	0.790	0.868	0.304	0.072	222.700	71120
SWAG	0.710	1.149	0.414	0.110	316.900	106745
PCA + ESS (SI)	0.817	0.679	0.263	0.038	196.600	136360
MC dropout	0.798	0.846	0.293	0.081	214.300	35560
SGD	0.806	0.785	0.280	0.046	205.100	17780

Table 3.16. Comparison of OOD detection performance while using WideResNet28x10 on CIFAR100.

Inference	OOD Dataset	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑
SGLD	STL10	0.791	0.819
	SVHN	0.790	0.786
SGHMC	STL10	0.788	0.824
	SVHN	0.760	0.741
cSGLD(50)	STL10	0.818	0.839
	SVHN	0.818	0.812
cSGHMC(50)	STL10	0.812	0.831
	SVHN	0.773	0.772
cSGLD(9)	STL10	0.804	0.829
	SVHN	0.784	0.782
cSGHMC(9)	STL10	0.792	0.817
	SVHN	0.772	0.792
SWAG	STL10	0.732	0.753
	SVHN	0.672	0.704
PCA + ESS (SI)	STL10	0.813	0.827
	SVHN	0.760	0.814
MC dropout	STL10	0.798	0.815
	SVHN	0.642	0.645
SGD	STL10	N/A	0.820
	SVHN	N/A	0.732

Table 3.17. Comparison of Misclassification detection while using WideResNet28x10 on CIFAR100.

Inference	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑	AUROC- Model Confidence ↑	AUCPR- Model Uncertainty ↑	AUCPR - Total Uncertainty ↑	AUCPR- Model Confidence ↑
SGLD	0.875	0.885	0.889	0.543	0.620	0.634
SGHMC	0.877	0.882	0.888	0.544	0.598	0.621
cSGLD(50)	0.887	0.888	0.895	0.567	0.591	0.613
cSGHMC(50)	0.882	0.882	0.890	0.589	0.596	0.625
cSGHMC(9)	0.864	0.877	0.880	0.575	0.629	0.646
cSGLD(9)	0.880	0.892	0.897	0.564	0.630	0.647
SWAG	0.837	0.857	0.860	0.601	0.675	0.686
PCA + ESS (SI)	0.853	0.868	0.888	0.520	0.559	0.603
MC dropout	0.883	0.887	0.887	0.617	0.638	0.637
SGD	N/A	0.869	0.879	N/A	0.586	0.622

to their non-cyclic versions. The relative performance trend that we observe here is very similar to that of the medium-scale benchmark. In terms of performance across the board, cSGHMC and cSGLD outperform the rest of the methods. Furthermore, cSGLD and cSGHMC perform very similar to their non-cyclic counterparts, which can be potentially attributed to the less number of samples or the number of cycles run. However, we notice here that PCA + ESS (SI) and SWAG do not perform as well as they did in the previous set of experiments. Finally, due to the high training runtime requirement of the cSGHMC/cSGLD, a user might yet again prefer SGHMC or MC Dropout as they provide reasonable performance in significantly less time.

Given the larger size of the ImageNet dataset (more than 1.2 million training images), we use multi-GPU training with a larger batch size of 256, processed using 24 PyTorch dataloader workers. For benchmarking the training time for large-scale benchmark, we use 4 Nvidia Titan X GPUs. We provide an approximate training time for large-scale benchmark results, using the same technique described in the previous subsection.

3.3.5 Edge Run Time Results

For the focus of this experiment, we look at the SGHMC ensemble, its distilled version, MC dropout, and the standard SGD point-estimated model. For the distillation experiment, the student model architecture matches that of the teacher model. Recall

Table 3.18. URSABench **large-scale** benchmark performance. For SGD, we use a pre-trained model available in PyTorch.

Inference	Accuracy ↑	NLL ↓	Robustness ↑	Uncertainty ↑	Training Time ↓
SGLD	0.768	0.927	0.832	0.851	23925
SGHMC	0.768	0.920	0.842	0.850	24243
cSGLD(6)	0.771	0.915	0.836	0.849	45675
cSGHMC(6)	0.771	0.910	0.838	0.849	46282
cSGLD(3)	0.769	0.920	0.830	0.848	28275
cSGHMC(3)	0.769	0.919	0.830	0.847	28650
SWAG	0.733	1.056	0.820	0.845	67022
PCA + ESS (SI)	0.765	0.948	0.813	0.847	110205
MC dropout	0.760	0.951	0.804	0.848	10825
SGD	0.760	0.962	-	-	-

Table 3.19. Comparison of predictive performance while using ResNet50 on ImageNet.

Inference	Accuracy ↑	NLL ↓	BS ↓	ECE ↓	Decision Cost ↓	Training Time
SGLD	0.768	0.927	0.326	0.034	6889	23925
SGHMC	0.768	0.920	0.325	0.028	6898	24243
cSGLD(6)	0.771	0.915	0.322	0.030	6773	45675
cSGHMC(6)	0.771	0.910	0.322	0.029	7078	46282
cSGLD(3)	0.769	0.920	0.324	0.032	7181	28275
cSGHMC(3)	0.769	0.919	0.324	0.031	7087	28650
SWAG	0.733	1.056	0.367	0.038	8089	67022
PCA + ESS (SI)	0.765	0.948	0.356	0.050	6782	110205
MC dropout	0.760	0.951	0.335	0.026	7152	10825
SGD	0.760	0.962	0.336	0.038	7431	-

Table 3.20. Comparison of OOD detection performance while using ResNet50 on ImageNet.

Inference	OOD Dataset	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑
SGLD	ImageNet-Sketch	0.828	0.837
SGHMC	ImageNet-Sketch	0.841	0.844
cSGLD(6)	ImageNet-Sketch	0.832	0.841
cSGHMC(6)	ImageNet-Sketch	0.833	0.843
cSGLD(3)	ImageNet-Sketch	0.822	0.837
cSGHMC(3)	ImageNet-Sketch	0.821	0.839
SWAG	ImageNet-Sketch	0.809	0.831
PCA + ESS (SI)	ImageNet-Sketch	0.800	0.827
MC dropout	ImageNet-Sketch	0.772	0.837
SGD	ImageNet-Sketch	N/A	0.835

Table 3.21. Comparison of Misclassification detection while using ResNet50 on ImageNet.

Inference	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑	AUROC- Model Confidence ↑	AUCPR- Model Uncertainty ↑	AUCPR - Total Uncertainty ↑	AUCPR- Model Confidence ↑
SGLD	0.829	0.857	0.867	0.547	0.616	0.649
SGHMC	0.826	0.856	0.867	0.531	0.616	0.649
cSGLD(6)	0.825	0.856	0.866	0.529	0.612	0.644
cSGHMC(6)	0.825	0.856	0.867	0.530	0.610	0.644
cSGLD(3)	0.818	0.858	0.867	0.521	0.615	0.647
cSGHMC(3)	0.817	0.857	0.867	0.518	0.612	0.645
SWAG	0.821	0.851	0.863	0.565	0.644	0.677
PCA + ESS (SI)	0.816	0.858	0.866	0.550	0.649	0.673
MC dropout	0.826	0.853	0.864	0.533	0.617	0.650
SGD	N/A	0.856	0.865	N/A	0.622	0.653

Table 3.22. URSABench edge performance-latency comparison. Here, # models indicate the maximum number of models runnable on the Jetson TX2 device for SGHMC. Ensm. size indicates the size of the original ensemble.

Dataset-Model	Inference	Accuracy ↑	NLL ↓	Robustness ↑	Uncertainty ↑	Latency ↓	# Models/Ensm. size
MNIST - MLP200	SGHMC	0.985	0.047	0.947	0.980	0.080	100/100
	MC dropout	0.984	0.048	0.945	0.980	0.017	1
	SGD	0.986	0.045	0.948	0.980	0.0006	1
	Distilled SGHMC	0.986	0.044	0.960	0.970	0.0006	1
CIFAR10 - ResNet50	SGHMC	0.949	0.255	0.765	0.929	0.352	49/50
	MC dropout	0.948	0.208	0.817	0.947	0.051	1
	SGD	0.943	0.274	0.787	0.937	0.007	1
	Distilled SGHMC	0.932	0.279	0.787	0.929	0.007	1
CIFAR10 - WideResNet28x10	SGHMC	0.966	0.140	0.809	0.943	0.154	12/50
	MC dropout	0.957	0.158	0.805	0.947	0.031	1
	SGD	0.963	0.138	0.815	0.941	0.013	1
	Distilled SGHMC	0.950	0.169	0.812	0.946	0.013	1
CIFAR100 - ResNet50	SGHMC	0.738	1.222	0.780	0.872	0.348	49/50
	MC dropout	0.786	1.006	0.776	0.878	0.071	1
	SGD	0.732	1.302	0.764	0.872	0.007	1
	Distilled SGHMC	0.732	1.088	0.794	0.871	0.007	1
CIFAR100 - WideResNet28x10	SGHMC	0.811	0.784	0.783	0.884	0.153	12/50
	MC dropout	0.798	0.846	0.730	0.887	0.053	1
	SGD	0.806	0.785	0.776	0.874	0.013	1
	Distilled SGHMC	0.785	0.839	0.812	0.877	0.013	1
ImageNet - ResNet50	SGHMC	0.768	0.920	0.844	0.862	0.085	6/6
	MC dropout	0.760	0.951	0.804	0.848	0.076	1
	SGD	0.760	0.962	0.835	0.861	0.014	1
	Distilled SGHMC	0.736	1.086	0.865	0.849	0.014	1

that MC dropout requires us to store a single model, and the posterior predictive distribution is computed using multiple forward passes. The 8GB physical memory of Jetson TX2 is shared by both the CPU and the GPU. As a result, we can only accommodate a certain number of models from the ensemble. We present the profiling results for the edge deployment are presented in Table 3.22.

As we can see, the number of model samples we can fit onto the Jetson TX2 varies by model type. Specifically, for WideResNet28x10, we can only fit 12 samples from the ensemble of 50 models. For ImageNet, the large model size means we were only able to deploy six samples onto the board. The results of SGHMC are computed using only the model samples that we could fit on the Jetson TX2. Additionally, for the point-estimated model and the distilled model, we excluded the model uncertainty from the uncertainty and robustness scores of the composite score, as model uncertainty requires an expectation over multiple samples of an ensemble.

For MNIST, we can clearly see that the performance of SGD or distilled SGHMC across all four composite scores is very similar to that of SGHMC and MC dropout, making them an attractive choice considering the latency values. For WideResNet28x10 on CIFAR10 dataset, we observe that SGD point-estimated models tend to perform better than its counterparts while looking at NLL, and have similar performance on other composite scores during comparison. It also has relatively very low latency. For ResNet50 on CIFAR datasets, the trade-off between different performance metrics and latency is high. MC dropout tends to perform best across all four composite scores against the other three approximate inference methods in this case, but it comes at a higher latency cost. Finally, for ImageNet, we observe that while SGHMC outperforms other methods across all the scores, the trade-off between latency and performance is not as significant when compared against SGD. In this case, a user might select the SGD point-estimated model for practical applications, with a small degradation in performance.

3.3.6 Discussion

We make several observations based on the benchmark results presented in this section. First, if we are primarily interested in the accuracy, robustness or uncertainty metrics, then cSGHMC and cSGLD (using more parameter samples) are often the best choice. Across the different benchmarks, we consistently see them outperforming other methods (see Table 3.1, 3.5, 3.18). Second, MC Dropout, SGHMC/SGLD and cSGHMC/cSGLD (using fewer parameter samples) can often provide reasonable uncertainty and accuracy performance for lower training time budgets. Third, PCA-based subspace inference and SWAG require more training time than SGLD and SGHMHC as can be seen in Tables 3.1, 3.5, and 3.18. This is largely due to the time needed to generate the subspace (for subspace inference) or the Gaussian approximation to the parameter posterior (for SWAG). Furthermore, PCA-based subspace inference and SWAG do not provide consistent performance improvement over the rest of the methods in terms of accuracy, NLL, robustness, and uncertainty based on these results.

For the large-scale benchmark, we also see that the SGD based point-estimated models are able to compete with the posterior ensemble approaches. This may be due to the limited number of MCMC samples used or greater relative difficulty in mixing and requires further study. For the edge deployment scenario considered, we see that MC dropout is a good alternative to SGHMC in terms of the tradeoff between prediction performance and prediction latency. It is also important to note that the other approximate inference techniques discussed in this chapter, apart from the ones in Table 3.22, have similar latency properties to SGHMC due to performing a forward pass for each of a set of parameter samples.

Finally, while the distillation approach is interesting, for our medium to large-scale benchmark setting, we observed that they do not perform any better than the SGD based models.

3.4 Conclusions

This chapter describes URSABench, a system for benchmarking multiple aspects of the performance of approximate Bayesian inference methods for deep neural networks. We hope that the development of this benchmarking system and its components will facilitate research in the domain of approximate Bayesian inference by better exposing the performance trade-offs achieved by different methods in terms of uncertainty, robustness, scalability and accuracy. We believe the simultaneous assessment of these properties is critical to better understand which methods are most effective on different downstream tasks and in different deployment contexts (e.g., server and edge).

A further challenge in the development of this system is ensuring a fair comparison between approaches. This can be difficult for new model/method/data set combinations without established hyperparameters, requiring careful hyperparameter optimization. This highlights the more complex issue of how to fairly benchmark the end-to-end process of hyperparameter optimization and inference in terms of computational resource use. In this work, the training time scalability measurements are assessed on tuned models only. The cost of hyperparameter optimization is not reflected, and thus methods with more hyperparameters that may take longer to optimize for new problems and tasks are not penalized in these results. This issue requires further study.

CHAPTER 4

GENERALIZED BAYESIAN POSTERIOR EXPECTATION DISTILLATION FOR DEEP NEURAL NETWORKS

In this chapter, we present a general framework for distilling expectations with respect to the Bayesian posterior distribution of a deep neural network classifier, extending prior work on the Bayesian Dark Knowledge framework. The proposed framework takes as input "teacher" and "student" model architectures and a general posterior expectation of interest. The distillation method performs an online compression of the selected posterior expectation using iteratively generated Monte Carlo samples. We focus on the posterior predictive distribution and expected entropy as distillation targets. We investigate several aspects of this framework, including the impact of uncertainty and the choice of student model architecture. We study methods for student model architecture search from a speed-storage-accuracy perspective and evaluate down-stream tasks leveraging entropy distillation including uncertainty ranking and out-of-distribution detection.

The remainder of the chapter is structured as follows. In Section 4.1, we present the proposed framework of distilling Bayesian posterior expectations. In Section 4.2, we present the experiments and results to evaluate our distillation framework. Finally, we provide a discussion and a set of potential future directions in Section 4.3 to conclude this chapter.

Bibliographical note: This chapter is adapted from Vadera et al. [2020b].

4.1 Framework

In this section, we describe our proposed framework.

4.1.1 Generalized Posterior Expectations

As described in the previous section, different statistics derived from the posterior distribution $p(\theta|\mathcal{D},\lambda)$ may be useful in different data analysis tasks. We consider the general case of inferences that take the form of posterior expectations as shown in Equation 4.1 where $g(y,\mathbf{x},\theta)$ is an arbitrary function of y, $\mathbf{x}$ and θ.

$$\mathbb{E}_{p(\theta|\mathcal{D},\lambda)}[g(y,\mathbf{x},\theta)] = \int p(\theta|\mathcal{D},\lambda)g(y,\mathbf{x},\theta)d\theta \tag{4.1}$$

Important examples of functions $g(y,\mathbf{x},\theta)$ include $g(y,\mathbf{x},\theta) = p(y|\mathbf{x},\theta)$, which results in the posterior predictive distribution $p(y|\mathbf{x},\mathcal{D},\lambda)$ as used in Bayesian Dark Knowledge. The choice $g(y,\mathbf{x},\theta) = \sum_{y'=1}^{C} p(y'|\mathbf{x},\theta)\log p(y'|\mathbf{x},\theta)$ yields the expected data uncertainty introduced in Chapter 2, Section 2.1. The choice $g(y,\mathbf{x},\theta) = p(y|\mathbf{x},\theta)(1-p(y|\mathbf{x},\theta))$ results in the posterior marginal variance of the class y given $\mathbf{x}$. We use the posterior predictive distribution and expected data uncertainty as examples throughout this work.

4.1.2 Generalized Posterior Expectation Distillation

Our goal is to learn to approximate posterior expectations $\mathbb{E}_{p(\theta|\mathcal{D},\lambda)}[g(y,\mathbf{x},\theta)]$ under a given teacher model architecture using a given student model architecture. The method that we propose takes as input the teacher model $p(y|\mathbf{x},\theta)$, the prior $p(\theta|\lambda)$, a labeled data set $\mathcal{D}$, an unlabeled data set $\mathcal{D}'$, the function $g(y,\mathbf{x},\theta)$, a student model $f(y,\mathbf{x}|\phi)$, an expectation estimator, and a loss function $\ell(\cdot,\cdot)$ that measures the error of the approximation given by the student model $f(y,\mathbf{x}|\phi)$.[1] Similar to Balan et al.

[1]Note that $f(y,\mathbf{x}|\phi)$ denotes the student's output probability for class y given input $\mathbf{x}$ and parameters ϕ.

Algorithm 1 Generalized Posterior Expectation Distillation (GPED)

```
    Input: $\mathcal{D}$, $\mathcal{D}'$, $p(y|\mathbf{x},\theta)$, $\lambda$, $g$, $f$, $U$, $\ell$, $R$, $M$, $M'$, $H$, $B$, $\lambda$, $\{\eta_t\}_{t=1}^{T}$, $\{\alpha_s\}_{s=1}^{S}$
 1: procedure GPED
 2:     Initialize $s = 0$, $\phi_0$, $\theta_0$, $\hat{g}_{yi0} = 0$, $m_{i0} = 0$,$\eta_0$
 3:     for $t = 0$ to $T$ do
 4:         Sample $\mathcal{S}$ from $\mathcal{D}$ with $|\mathcal{S}| = M$
 5:         $\theta_{t+1} \leftarrow \theta_t + \frac{\eta_t}{2}\Big(\nabla_\theta \log p(\theta|\lambda) + \frac{N}{M}\sum_{i\in\mathcal{S}} \nabla_\theta \log p\left(y_i|x_i,\theta_t\right)\Big) + z_t$
 6:         if $\mod(t,H) = 0$ and $t > B$ then
 7:             Sample $\mathcal{S}'$ from $\mathcal{D}'$ with $|\mathcal{S}'| = M'$
 8:             for $i \in \mathcal{S}'$ do
 9:                 $\hat{g}_{yis+1} \leftarrow U(\hat{g}_{yis}, \theta_t, m_{is})$
10:                 $m_{is+1} \leftarrow m_{is} + 1$
11:             end for
12:             $\phi_{s+1} \leftarrow \phi_s + \alpha_s\Big(\frac{N'}{M'}\sum_{i\in\mathcal{S}'}\sum_{y\in\mathcal{Y}} \nabla_\phi \ell\big(\hat{g}_{yis+1}, f(y,\mathbf{x}_i|\phi_s)\big) + \lambda\nabla_\phi R(\phi_s)\Big)$
13:             $s \leftarrow s + 1$
14:         end if
15:     end for
16: end procedure
```

[2015], we propose an online distillation method based on the use of the SGLD sampler. We describe all the components of the framework in the sections below, and provide a complete description of the resulting method in Algorithm 1.

SGLD Sampler: The prior distribution over the parameters $p(\theta|\lambda)$ is chosen to be a spherical Gaussian distribution with mean $\mu = 0$ and precision τ (we thus have $\lambda = [\mu, \tau]$). We define $\mathcal{S}$ to be a minibatch of size M drawn from $\mathcal{D}$. θ_t denotes the parameter set sampled for the teacher model at sampling iteration t, while η_t denotes the step size for the teacher model at iteration t. The Langevin noise is denoted by $z_t \sim \mathcal{N}(0, \eta_t I)$. The sampling update for SGLD is given b: $\theta_{t+1} \leftarrow \theta_t + \Delta\theta_t$ where $\Delta\theta_t$ is defined as:

$$\Delta\theta_t = \frac{\eta_t}{2}\left(\nabla_\theta \log p(\theta|\lambda) + \frac{N}{M}\sum_{i\in\mathcal{S}} \nabla_\theta \log p\left(y_i|x_i,\theta_t\right)\right) + z_t \tag{4.2}$$

Distillation Procedure: For the distillation learning procedure, we make use of a secondary unlabeled data set $\mathcal{D}' = \{\mathbf{x}_i | 1 \le i \le N'\}$. This data set could use feature vectors from the primary data set $\mathcal{D}$, or a larger data set. We note that due to autocorrelation in the sampled teacher model parameters θ_t, we may not want to run a distillation update for every Monte Carlo sample drawn. We thus use two different iteration indices: t for SGLD iterations and s for distillation iterations.

On every distillation step s, we sample a minibatch $\mathcal{S}'$ from $\mathcal{D}'$ of size M'. For every data case i in $\mathcal{S}'$, we update an estimate $\hat{g}_{yis}$ of the posterior expectation using the most recent parameter sample θ_t, obtaining an updated estimate $\hat{g}_{yis+1} \approx \mathbb{E}_{p(\theta|\mathcal{D},\lambda)}[g(y, \mathbf{x}_i, \theta)]$ (we discuss update schemes in the next section). Next, we use the minibatch of examples $\mathcal{S}'$ to update the student model. To do so, we take a step $\phi_{s+1} \leftarrow \phi_s + \alpha_s \Delta\phi_s$ in the gradient direction of the regularized empirical risk of the student model as shown below where α_s is the student model learning rate at step s, $R(\phi)$ is the regularizer, and λ is the regularization hyperparameter. We next discuss the estimation of the expectation targets $\hat{g}_{yis}$.

$$\Delta\phi_s = \frac{N'}{M'} \sum_{i \in \mathcal{S}'} \sum_{y \in \mathcal{Y}} \nabla_\phi \ell\big(\hat{g}_{yis+1}, f(y, \mathbf{x}_i|\phi_s)\big) + \lambda \nabla_\phi R(\phi_s) \tag{4.3}$$

Expectation Estimation: Given an explicit collection of posterior samples $\theta_1, ..., \theta_s$, the standard Monte Carlo estimate of $\mathbb{E}_{p(\theta|\mathcal{D},\lambda)}[g(y, \mathbf{x}, \theta)]$ is simply $\hat{g}_{yis} = \frac{1}{S}\sum_{j=1}^{s} g(y, \mathbf{x}_i, \theta_j)$. However, this estimator requires retaining the sequence of samples $\theta_1, ..., \theta_s$, which may not be feasible in terms of storage cost. Instead, we consider the application of an online update function. We define m_{is} to be the count of the number of times data case i has been sampled up to and including distillation iteration s. An online update function $U(\hat{g}_{yis}, \theta_t, m_{is})$ takes as input the current estimate of the expectation, the current sample of the model parameters, and the number of times data case i has been sampled, and produces an updated estimate of the expectation $\hat{g}_{yis+1}$. Below, we define two different versions of the function. $U_s(\hat{g}_{yis}, \theta_t, m_{is})$, updates $\hat{g}_{yis}$ using the current sample only, while $U_o(\hat{g}_{yis}, \theta_t, m_{is})$ performs an online update equivalent to a full Monte Carlo average.

$$U_s(\hat{g}_{yis}, \theta_t, m_{is}) = g(y, \mathbf{x}_i, \theta_t) \tag{4.4}$$

$$U_o(\hat{g}_{yis}, \theta_t, m_{is}) = \frac{1}{m_{is+1}}\big(m_{is} \cdot \hat{g}_{yis} + g(y, \mathbf{x}_i, \theta_t)\big) \tag{4.5}$$

We note that both update functions provide unbiased estimates of $\mathbb{E}_{p(\theta|\mathcal{D},\lambda)}[g(y, \mathbf{x}, \theta)]$ after a suitable burn-in time B. The online update $U_o(.)$ will generally result in lower variance in the estimated values of $\hat{g}_{yis}$, but it comes at the cost of needing to explicitly maintain the expectation estimates $\hat{g}_{yis}$ across learning iterations, increasing the storage cost of the algorithm. It is worthwhile noting that the extra storage and computation cost required by U_o grows linearly in the size of the training set for the student. By contrast, the fully stochastic update is memory-less in terms of past expectation estimates, so the estimated expectations $\hat{g}_{yis}$ do not need to be retained across iterations, resulting in a substantial space savings.

General Algorithm and Special Cases: We show a complete description of the proposed method in Algorithm 1. The algorithm takes as input the teacher model $p(y|\mathbf{x}, \theta)$, the parameters of the prior $p(\theta|\lambda)$, a labeled data set $\mathcal{D}$, an unlabeled data set $\mathcal{D}'$, the function $g(y, \mathbf{x}, \theta)$, the student model $f(y, \mathbf{x}|\phi)$, an online expectation estimator $U(\hat{g}_{yis}, \theta_t, m_{is})$, a loss function $\ell(\cdot, \cdot)$ that measures the error of the approximation given by $f(y, \mathbf{x}|\phi)$, a regularization function $R(.)$ and regularization hyper-parameter λ, minibatch sizes M and M', the thinning interval parameter H, the SGLD burn-in time parameter B and step size schedules for the step sizes η_t and α_s.

We note that the original Bayesian Dark Knowledge method is recoverable as a special case of this framework via the the choices $g(y, \mathbf{x}, \theta) = p(y|\mathbf{x}, \theta)$, $\ell(p, q) = -p\log(q)$, $U = U_s$ and $p(y|\mathbf{x}, \theta) = f(y, \mathbf{x}, \phi)$ (i.e., the architecture of the student is selected to match that of the teacher). The original approach also uses a distillation data set $\mathcal{D}'$ obtained from $\mathcal{D}$ by adding randomly generated noise to instances from $\mathcal{D}$ on each distillation iteration, taking advantage of the fact that the choice $U = U_s$ means that no aspect of the algorithm scales with $|\mathcal{D}'|$.

Our general framework allows for other trade-offs, including reducing the variance in the estimates of $\hat{g}_{yis}$ at the cost of additional storage in proportion to $|\mathcal{D}'|$. We also note that the loss function $\ell(p,q) = -p\log(q)$ and the choice $g(y,\mathbf{x},\theta) = p(y|\mathbf{x},\theta)$ are somewhat of a special case when used together as even when the full stochastic expectation update U_s is used, the resulting distillation parameter gradient is unbiased. To distill posterior expected entropy (e.g., expected data uncertainty), we set $g(y,\mathbf{x},\theta) = \sum_{y\in\mathcal{Y}} p(y|\mathbf{x},\theta)\log p(y|\mathbf{x},\theta)$, $U = U_o$ and $\ell(h,h') = |h-h'|$.

4.1.3 Model Compression and Pruning

One of the primary motivations for the original Bayesian Dark Knowledge approach is that it provides an approximate inference framework that results in significant computational and storage savings at test time. However, a drawback of the original approach is that the architecture of the student is chosen to match that of the teacher. As we will show in Section 4.2, this will sometimes result in a student network that has too little capacity. On the other hand, if we plan to deploy the student model in a low resource compute environment, the teacher architecture may not meet the specified computational constraints. In either case, we need a general approach for selecting an architecture for the student model.

To begin to explore this problem, we consider two basic approaches to choosing student model architectures that enable trading off test time inference speed and storage for accuracy (or more generally, lower distillation loss). A helpful aspect of the distillation process relative to a de novo architecture search problem is that the architecture of the teacher model is available as a starting point. As a first approach, we consider wrapping the proposed GPED algorithm with an explicit search over a set of student models that are "close" to the teacher. Specifically, we consider a search space obtained by starting from the teacher model and applying a width multiplier to the width of every fully connected layer and a kernel multiplier to the number of kernels in every convolutional layer. While this search requires exponential time in the number of layers, it provides a baseline for evaluating other methods.

As an alternative approach with better computational complexity, we leverage the regularization function $R(\phi)$ included in the GPED framework to prune a large initial network using group ℓ_1/ℓ_2 regularization [Zhang and Ou, 2018, Wen et al., 2016]. To apply this approach, we first must partition the parameters in the parameter vector ϕ across K groups $\mathcal{G}_k$. The form of the regularizer is $R(\phi) = \sum_{k=1}^{K} \left(\sum_{j \in \mathcal{G}_k} \phi_j^2 \right)^{1/2}$. As is well-established in the literature, this regularizer causes all parameters in a group to go to zero simultaneously when they are not needed in a model. To use it for model pruning for a unit in a fully connected layer, we collect all of that unit's inputs into a group. Similarly, we collect all the incoming weights for a particular channel in a convolution layer together into a group. If all incoming weights associated with a unit or a channel have magnitude below a small threshold ϵ, we can explicitly remove them from the model, obtaining a more compact architecture. We also fine-tune our models after pruning.

Finally, we note that any number of weight compressing, pruning, and architecture search methods could be combined with the GPED framework. Our goal is not to exhaustively compare such methods, but rather to demonstrate that GPED is sensitive to the choice of student model to highlight the need for additional research on the problem of selecting student model architectures.

4.2 Experiments and Results

In this section, we present experiments and results evaluating the proposed approach using multiple data sets, posterior expectations, teacher model architectures, student model architectures, basic architecture search methods, and multiple down-stream tasks. We begin by providing an overview of the experimental protocols used.

4.2.1 Experimental Protocols

4.2.1.1 Datasets

We use the MNIST [LeCun, 1998] and CIFAR10 [Krizhevsky et al., 2009] data sets as base data sets in our experiments. The original empirical investigation of Bayesian Dark Knowledge for classification focused on the MNIST data set [LeCun, 1998]. However, the models fit to the MNIST data set have very low posterior uncertainty, and we argue that it is thus a poor benchmark for assessing the performance of posterior distillation methods. In this section, we investigate two orthogonal modifications of the standard MNIST data set to increase uncertainty: reducing the training set size and masking regions of the input images. Our goal is to produce a range of benchmark problems with varying posterior predictive uncertainty. We also use the CIFAR10 data set [Krizhevsky et al., 2009] in our experiments and employ the same subsampling technique.

MNIST: The full MNIST dataset consists of 60,000 training images and 10,000 test images, each of size 28×28, distributed among 10 classes LeCun [1998]. As a first manipulation, we consider sub-sampling the labeled training data to include 10,000, 20,000, 30,000 or all 60,000 data cases in the primary data set $\mathcal{D}$ when performing posterior sampling for the teacher model. Importantly, we use all 60,000 unlabeled training cases in the distillation data set $\mathcal{D}'$. This allows us decouple the impact of reduced labeled training data on posterior predictive distributions from the effect of the amount of unlabeled data available for distillation.

As a second manipulation, we generate images with occlusions by randomly masking out parts of each available training and test image. For generating such images, we randomly choose a square $m \times m$ region (mask) and set the value for pixels in that region to 0. Thus, the masking rate for a 28×28 MNIST image corresponding to the mask of size $m \times m$ is given by $r = \frac{m \times m}{28 \times 28}$. We illustrate original and masked data in

Figure 4.1. We consider a range of square masks, resulting in masking rates between 0% and 86.2%.

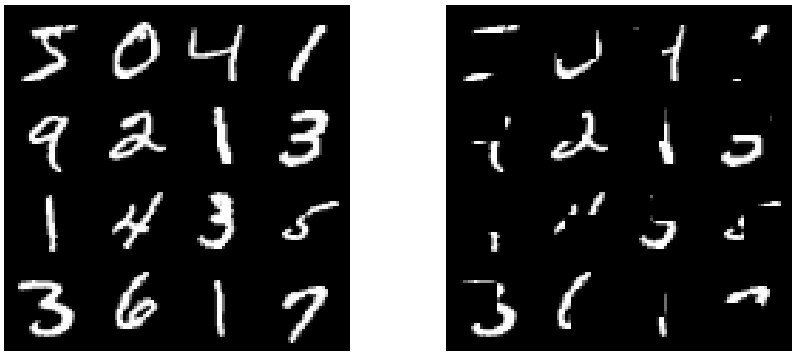

(a) Original images (b) Processed images

Figure 4.1. Example MNIST data after masking with $m = 14$.

CIFAR10: The full CIFAR10 dataset consists of 50,000 training images and 10,000 test images, each of size 32×32 pixels. We sub-sample the data into a primary training sets $\mathcal{D}$ containing 10,000, 20,000, and 50,000 images. As with MNIST, the sub-sampling is limited to training the teacher model only and we utilize all the 50,000 unlabeled training images in the distillation data set $\mathcal{D}'$.

4.2.1.2 Models

To demonstrate the generalizability of our methods to a range of model architectures, we run our experiments with both fully-connected, and convolutional neural networks. We evaluate a total of three teacher models in this work: a three-layer fully connected network (FCNN) for MNIST matching the architecture used by Balan et al. [2015], a four-layer convolutional network for MNIST, and a five-layer convolutional network for CIFAR10. We note that our goal in this work is not to evaluate the GPED framework on state-of-the-art architectures, but rather to provide illustrative results and establish methodology for assessing the impact of several factors including the level of uncertainty and the architecture of the student model.

Teacher Models: We begin by defining the architectures used for the teacher model as follows:

1. **FCNN (MNIST)**: We use a 3-layer fully connected neural network. The architecture used is: Input(784)-FC(400)-FC(400)-FC(output). This matches the architecture used by Balan et al. [2015].
2. **CNN (MNIST)**: For a CNN, we use two consecutive sets of 2D convolution and max-pooling layers, followed by two fully-connected layers. The architecture used is: Input(1, (28,28))-Conv(num_kernels=10, kernel_size=4, stride=1) - MaxPool(kernel_size=2) - Conv(num_kernels=20, kernel_size=4, stride=1) - MaxPool(kernel_size=2) - FC (80) - FC (output).
3. **CNN (CIFAR10)**: Similar to the CNN architecture used for MNIST, we use two consecutive sets of 2D convolution and max-pooling layers followed by fully-connected layers. Conv(num_kernels=16, kernel_size=5) - MaxPool(kernel_size=2) - Conv(num_kernels=32, kernel_size=5) - MaxPool(kernel_size=2) - FC(200) - FC (50) - FC (output).

In the architectures mentioned above, the "output" size will change depending on the expectation that we're distilling. For classification, the output size will be 10 for both datasets, while for the case of entropy, it will be 1. We use *ReLU* non-linearities everywhere between the hidden layers. For the final output layer, *softmax* is used for classification. In the case of entropy, we use an exponential activiation to ensure positivity.

Student Models: The student models used in our experiments use the above mentioned architectures as the base architecture. For explicitly searching the space of the student models, we use a set of width multipliers starting from the teacher architecture. The space of student architectures corresponding to each teacher model defined earlier is given below. The width multiplier values of K_1 and K_2 are determined differently for each of the experiments, and thus will be mentioned in later sections.

1. **FCNN (MNIST)**: Input(784)-FC($400 \cdot K_1$)-FC($400 \cdot K_2$)-FC(output).
2. **CNN (MNIST)**: Input(1, (28,28))-Conv(num_kernels=$\lfloor 10 \cdot K_1 \rfloor$, kernel_size=4, stride=1) - MaxPool(kernel_size=2) - Conv(num_kernels=$\lfloor 20 \cdot K_1 \rfloor$, kernel_size=4, stride=1) - MaxPool(kernel_size=2) - FC ($\lfloor 80 \cdot K_2 \rfloor$) - FC (output).
3. **CNN (CIFAR10)**: Input(3, (32,32))-Conv(num_kernels=$\lfloor 16 \cdot K_1 \rfloor$, kernel_size=5) - MaxPool(kernel_size=2) - Conv(num_kernels=$\lfloor 16 \cdot K_1 \rfloor$, kernel_size=5) - MaxPool(kernel_size=2) - FC ($\lfloor 200 \cdot K_2 \rfloor$) - FC ($\lfloor 50 \cdot K_2 \rfloor$) - FC (output).

Distillation Procedures: We consider distilling both the posterior predictive distribution and the posterior entropy, as described in the previous section. For the posterior predictive distribution, we use the stochastic expectation estimator, U_s while for entropy we experiment with both estimators. We run the distillation procedure using the following hyperparameters: fixed teacher learning rate $\eta_t = 4 \times 10^{-6}$ for models on MNIST and $\eta_t = 2 \times 10^{-6}$ for models on CIFAR10, teacher prior precision $\tau = 10$, initial student learning rate $\alpha_s = 10^{-3}$, student dropout rate $p = 0.5$ for fully-connected models on MNIST (and zero otherwise), burn-in iterations $B = 1000$ for MNIST and $B = 10000$ for CIFAR10, thinning interval $H = 100$ for distilling predictive means and $H = 10$ for distilling entropy values, and total training iterations $T = 10^6$. For training the student model, we use the *Adam* algorithm (instead of plain steepest descent as indicated in Algorithm 1) and set a learning schedule for the student such that it halves its learning rate every 200 epochs for models on MNIST, and every 400 epochs for models on CIFAR10. Also, note that we only apply the regularization function $R(\phi_s)$ while doing Group ℓ_1/ℓ_2 pruning. Otherwise, we use dropout as indicated before.

Table 4.1. Results of posterior distillation when the student architecture is fixed to match the teacher architecture and base data sets are used with no sub-sampling or occlusion.

Model & Dataset	Teacher NLL	Student NLL	MAE (Entropy)
FCNN - MNIST	0.052	0.082	0.016
CNN - MNIST	0.022	0.053	0.016
CNN - CIFAR10	0.671	0.932	0.245

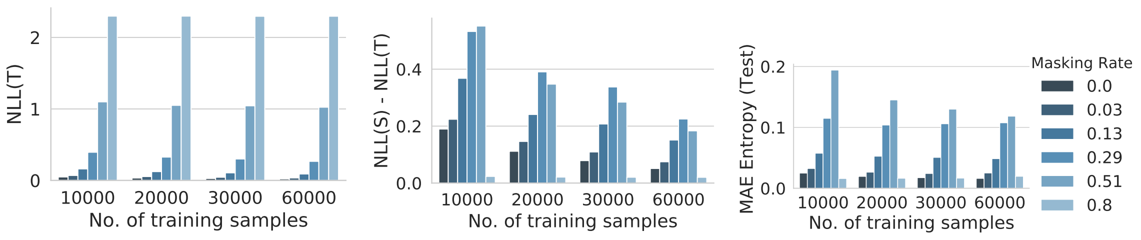
4.2.2 Experiments

4.2.2.1 Experiment 1: Distilling Posterior Expectations

For this experiment, we use the MNIST and CIFAR10 datasets without any subsampling or masking. For each dataset and model, we consider separately distilling the posterior predictive distribution and the posterior entropy. We fix the architecture of the student to match that of the teacher. To evaluate the performance while distilling the posterior predictive distribution, we use the negative log-likelihood (NLL) of the model on the test set. For evaluating the performance of distilling posterior entropy, we use the mean absolute difference between the teacher ensemble's entropy estimate and the student model output on the test set. The results are given in Table 4.1. First, we note that the FCNN NLL results on MNIST closely replicate the results in Balan et al. [2015], as expected. We also note that the error in the entropy is low for both the FCNN and CNN architectures on MNIST. However, the student model fails to match the NLL of the teacher on CIFAR10 and the entropy MAE is also relatively high. In Experiment 2, we will investigate the effect of increasing uncertainty, while in Experiment 3 we will investigate the impact of student architectures.

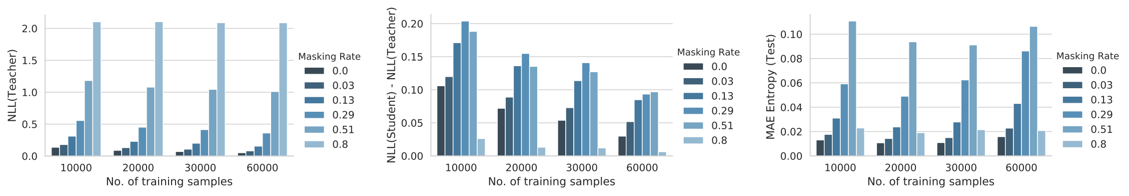
4.2.2.2 Experiment 2: Robustness to Uncertainty

We build on Experiment 1 by exploring methods for increasing posterior uncertainty on MNIST (sub-sampling and masking) and CIFAR10 (sub-sampling). We consider the cross product of four sub-sampling rates and six masking rates for MNIST and three



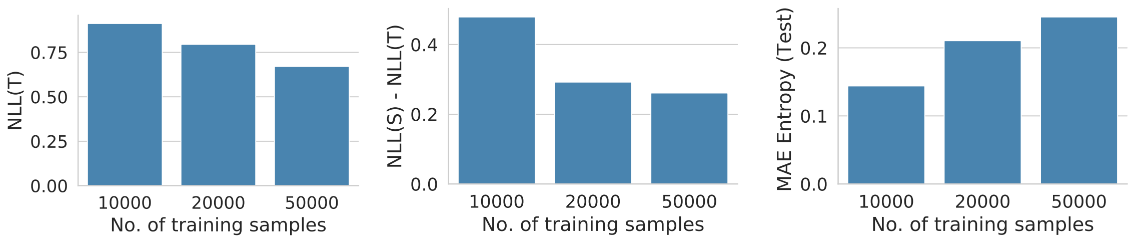
(a) (b) (c)

Figure 4.2. Distillation performance using CNNs on MNIST while varying data set size and masking rate. (a) Test negative log likelihood of the teacher posterior predictive distribution. (b) Difference in test negative log likelihood between student and teacher posterior predictive distribution estimates. (c) Difference between teacher and student posterior entropy estimates on test data set.



(a) (b) (c)

Figure 4.3. Distillation performance using Fully-Connected Networks on MNIST while varying data set size and masking rate. (a) Test negative log likelihood of the teacher posterior predictive distribution. (b) Difference in test negative log likelihood between teacher and student posterior predictive distribution estimates. (c) Difference between teacher and student posterior entropy estimates on test data set.



(a) (b) (c)

Figure 4.4. Distillation performance using CNNs on CIFAR10 while varying data set size. (d) Test negative log likelihood of the teacher posterior predictive distribution. (e) Difference in test negative log likelihood between student and teacher posterior predictive distribution estimates. (f) Difference between teacher and student posterior entropy estimates on test data set. In the plots above, S denotes the student and T denotes the teacher.

sub-sampling rates for CIFAR10. We consider the posterior predictive distribution and posterior entropy distillation targets. For the posterior predictive distribution we report the negative log likelihood (NLL) of the teacher, and the NLL gap between the teacher and student. For entropy, we report the mean absolute error between the teacher ensemble and the student. All metrics are evaluated on held-out test data. We also restrict the experiment to the case where the student architecture matches the teacher architecture, mirroring the Bayesian Dark Knowledge approach. In Figure 4.2, we show the results for the convolutional models on MNIST. The FCNN results are similar to the CNN results on MNIST and are shown in Figure 4.3 along with the CNN results on CIFAR10 in Figure 4.4. We also provide a performance comparison between the U_o and U_s estimators while distilling posterior expectations in Table 4.2.

As expected, the NLL of the teacher decreases as the data set size increases. We observe that changing the number of training samples has a similar effect on NLL gap for both CIFAR10 and MNIST. More specifically, for any fixed masking rate of MNIST (and zero masking rate for CIFAR10), we can see that the NLL difference between the student and teacher decreases with increasing training data. However, for MNIST we can see that the teacher NLL increases much more rapidly as a function of the masking rate. Moreover, the gap between the teacher and student peaks for moderate values of the masking rate. This fact is explained through the observation that when the masking rate is low, posterior uncertainty is low, and distillation is relatively easy. On the other hand, when the masking rate is high, the teacher essentially outputs the uniform distribution for every example, which is very easy for the student to represent. As a result, the moderate values of the masking rate result in the hardest distillation problem and thus the largest performance gap. For varying masking rates, we see exactly the same trend for the gap in posterior entropy predictions on MNIST. However, the gap for entropy prediction increases as a function of data set size for

CIFAR10. Finally, as we would expect, the performance of distillation using the U_o estimator is almost always better than that of the U_s estimator (see Table 4.2).

The key finding of this experiment is that the quality of the approximations provided by the student model can significantly vary as a function of properties of the underlying data set. In the next experiment, we address the problem of searching for improved student model architectures.

Table 4.2. Performance comparison between U_o and U_s estimators for convolutional neural network on MNIST. The NLL results correspond to the case of distilling the posterior predictive distribution, while the MAE on entropy results correspond to the case of distilling the expectation of predictive entropy.

Num. training samples	Masking rate	NLL (Teacher)	NLL (Student, U_o)	NLL (Student, U_s)	MAE (Entropy, U_o)	MAE (Entropy, U_s)
10000	0	0.048	0.214	0.218	0.025	0.030
	0.03	0.069	0.274	0.274	0.033	0.038
	0.13	0.161	0.509	0.509	0.058	0.069
	0.29	0.394	0.902	0.907	0.115	0.129
	0.51	1.099	1.615	1.630	0.194	0.170
	0.8	2.298	2.301	2.301	0.016	0.019
20000	0	0.034	0.126	0.126	0.020	0.021
	0.03	0.054	0.180	0.181	0.026	0.030
	0.13	0.123	0.342	0.344	0.053	0.066
	0.29	0.326	0.684	0.697	0.104	0.122
	0.51	1.050	1.369	1.378	0.145	0.150
	0.8	2.298	2.300	2.299	0.016	0.020
30000	0	0.028	0.084	0.086	0.017	0.019
	0.03	0.044	0.132	0.134	0.024	0.027
	0.13	0.106	0.292	0.294	0.051	0.061
	0.29	0.300	0.620	0.618	0.106	0.120
	0.51	1.044	1.307	1.308	0.130	0.141
	0.8	2.296	2.297	2.296	0.017	0.021
60000	0	0.022	0.053	0.053	0.016	0.017
	0.03	0.035	0.088	0.090	0.025	0.026
	0.13	0.090	0.219	0.221	0.049	0.058
	0.29	0.267	0.463	0.472	0.108	0.120
	0.51	1.024	1.184	1.187	0.118	0.127
	0.8	2.297	2.297	2.297	0.020	0.023

4.2.2.3 Experiment 3: Student Model Architectures

In this experiment, we compare exhaustive search to the group ℓ_1/ℓ_2 (group lasso) regularizer combined with pruning. For the pruning approach, we start with the largest

Table 4.3. Performance comparison between U_o and U_s estimators for fully-connected network on MNIST. The NLL results correspond to the case of distilling the posterior predictive distribution while the MAE on entropy results correspond to the case of distilling the expectation of predictive entropy.

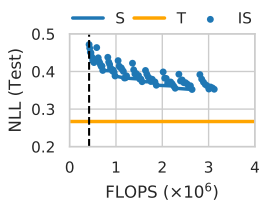
Num. training samples	Masking rate	NLL (Teacher)	NLL (Student, U_o)	NLL (Student, U_s)	MAE (Entropy, U_o)	MAE (Entropy, U_s)
10000	0	0.137	0.184	0.243	0.013	0.018
	0.03	0.180	0.233	0.300	0.018	0.023
	0.13	0.312	0.389	0.483	0.031	0.040
	0.29	0.556	0.637	0.760	0.059	0.089
	0.51	1.183	1.229	1.371	0.111	0.135
	0.8	2.103	2.111	2.129	0.023	0.019
20000	0	0.089	0.115	0.161	0.011	0.015
	0.03	0.131	0.165	0.220	0.014	0.021
	0.13	0.230	0.280	0.366	0.024	0.042
	0.29	0.452	0.510	0.607	0.049	0.104
	0.51	1.080	1.120	1.215	0.094	0.112
	0.8	2.104	2.108	2.117	0.019	0.021
30000	0	0.071	0.083	0.124	0.011	0.014
	0.03	0.107	0.129	0.180	0.015	0.021
	0.13	0.201	0.243	0.314	0.028	0.052
	0.29	0.414	0.459	0.555	0.062	0.105
	0.51	1.044	1.082	1.172	0.091	0.105
	0.8	2.089	2.092	2.101	0.022	0.023
60000	0	0.052	0.054	0.082	0.016	0.020
	0.03	0.081	0.094	0.133	0.023	0.034
	0.13	0.155	0.186	0.240	0.043	0.068
	0.29	0.360	0.398	0.471	0.086	0.109
	0.51	1.010	1.033	1.107	0.106	0.099
	0.8	2.088	2.089	2.094	0.021	0.022

Table 4.4. Performance comparison between U_o and U_s estimators for convolutional neural network on CIFAR10. The NLL results correspond to the case of distilling the posterior predictive distribution while the MAE on entropy results correspond to the case of distilling the expectation of predictive entropy.

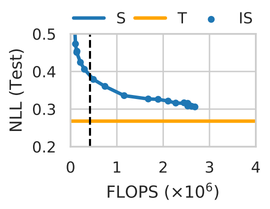
Num. training samples	NLL (Teacher)	NLL (Student, U_o)	NLL (Student, U_s)	MAE (Entropy, U_o)	MAE (Entropy, U_s)
10000	0.912	1.372	1.391	0.144	0.192
20000	0.798	1.184	1.179	0.210	0.231
50000	0.671	0.924	0.932	0.245	0.290

student model considered under exhaustive search, and prune back from there using different regularization parameters λ, leading to different student model architectures. We present results in terms of performance versus computation time (estimated in FLOPS), as well as performance vs storage cost (estimated in number of parameters). As performance measures for the posterior predictive distribution, we consider accuracy and negative log likelihood. For entropy, we use mean absolute error. In all cases, results are reported on test data. We consider both fully connected and convolutional models.

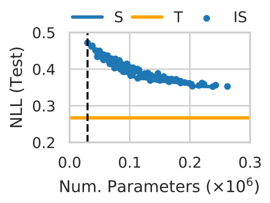
Figure 4.5 shows results for the negative log likelihood (NLL) of the convolutional model on MNIST with masking rate 29% and 60,000 training samples. We select this setting as illustrative of a difficult case for posterior predictive distribution distillation. We plot NLL vs FLOPS and NLL vs storage for all points encountered in each search. The solid blue line indicates the Pareto frontier.



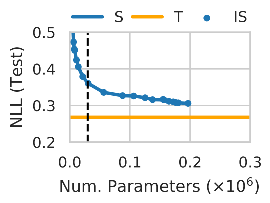
(a)



(b)



(c)

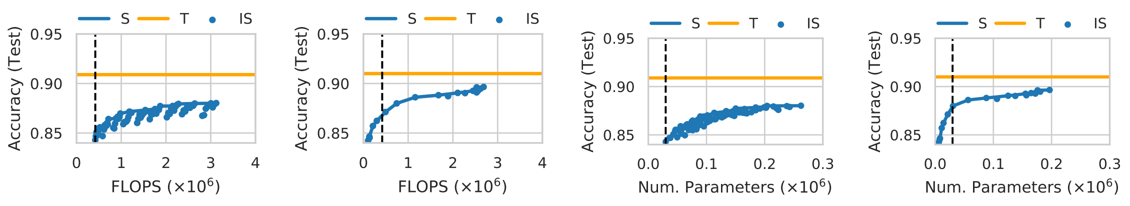


(d)

Figure 4.5. NLL-Storage-Computation tradeoff while using CNNs on MNIST with masking rate 29%. Test negative log likelihood of posterior predictive distribution vs FLOPS found using (a) exhaustive search and (b) group ℓ_1/ℓ_2 with pruning. Test negative log likelihood of posterior predictive distribution vs storage found using (c) exhaustive search and (d) group ℓ_1/ℓ_2 with pruning. The optimal student model for this configuration is obtained with group ℓ_1/ℓ_2 pruning. It has approximately 6.6$\times$ the number of parameters and 6.4$\times$ the FLOPS of the base student model. Notation: "S" - pareto frontier of the student models, "T" - Teacher, "IS" - Individual Student. The black dashed line denotes the FLOPS/number of parameters of the base student model having the same architecture as a teacher model.

First, we note that the baseline student model (with architecture matching the teacher) from Experiment 2 on MNIST achieves an NLL of 0.469 at approximately 0.48×10^6

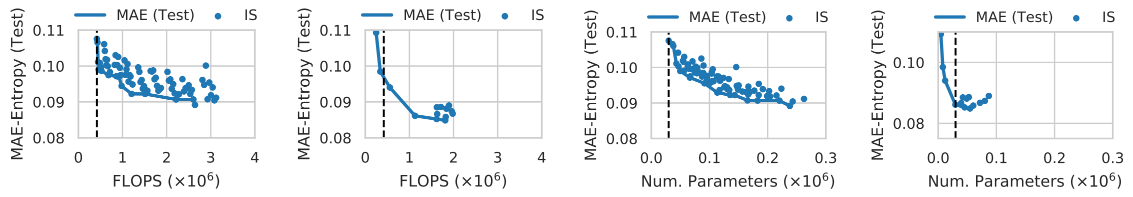
FLOPs and 0.03×10^6 parameters on this configuration of the data set. We can see that both methods for selecting student architectures provide a highly significant improvement over the baseline student architectures. On MNIST, the NLL is reduced to 0.30. Further, we can also see that the group ℓ_1/ℓ_2 approach is able to obtain much better NLL at the same computation and storage cost relative to the exhaustive search method. Lastly, the group ℓ_1/ℓ_2 method is able to obtain models on MNIST at less than 50% the computational cost needed by the baseline model with only a small loss in performance. The additional results from running Experiment 3 (Section 4.4) on different combinations of model type, dataset, and performance metrics have been given in Figures[4.6 - 4.13].



(a) (b) (c) (d)

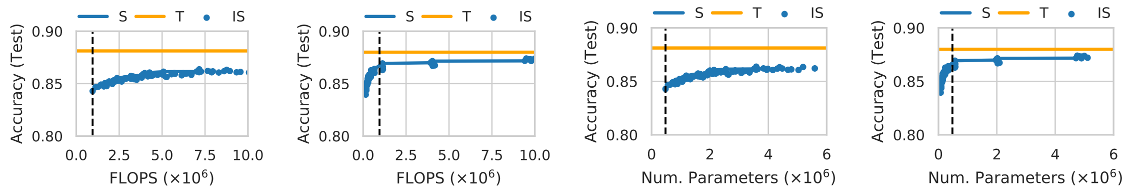
Figure 4.6. Accuracy-Storage-Computation tradeoff while using CNNs on MNIST with masking rate 29%. (a) Test accuracy using posterior predictive distribution vs FLOPS found using exhaustive search. (b) Test accuracy using posterior predictive distribution vs FLOPS found using group ℓ_1/ℓ_2 with pruning. (c) Test accuracy using posterior predictive distribution vs storage found using exhaustive search. (d) Test accuracy using posterior predictive distribution vs storage found using group ℓ_1/ℓ_2 with pruning. The optimal student model for this configuration is obtained with group ℓ_1/ℓ_2 pruning. It has approximately 6.6× the number of parameters and 6.4× the FLOPS of the base student model. Notation: "S" - pareto frontier of the student models, "T" - Teacher, "IS" - Individual Student. The black dashed line denotes the FLOPS/no. of parameters of the base student model having the same architecture as a teacher model.

In summary, the key finding of this experiment is that the capacity of the student model significantly impacts distillation performance, and student model architecture optimization methods are needed to achieve a desired speed-storage-accuracy trade-off.



(a) (b) (c) (d)

Figure 4.7. Entropy Error-Storage-Computation tradeoff while using CNNs on MNIST with masking rate 29%. (a) Test mean absolute error for posterior entropy vs FLOPS found using exhaustive search. (b) Test mean absolute error for posterior entropy vs FLOPS found using group ℓ_1/ℓ_2 with pruning. (c) Test mean absolute error for posterior entropy vs storage found using exhaustive search. (d) Test mean absolute error for posterior entropy vs storage found using group ℓ_1/ℓ_2 with pruning. The optimal student model for this configuration is obtained with group ℓ_1/ℓ_2 pruning. It has approximately 1.8× the number of parameters and 4.3× the FLOPS of the base student model. Notation: "MAE (Test)" - pareto frontier of the MAEs obtained using different student models, "IS" - Individual Student. The black dashed line denotes the FLOPS/no. of parameters of the base student model having the same architecture as a teacher model.



(a) (b) (c) (d)

Figure 4.8. Accuracy-Storage-Computation tradeoff while using Fully-connected networks on MNIST with masking rate 29%. (a) Test accuracy using posterior predictive distribution vs FLOPS found using exhaustive search. (b) Test accuracy using posterior predictive distribution vs FLOPS found using group ℓ_1/ℓ_2 with pruning. (c) Test accuracy using posterior predictive distribution vs storage found using exhaustive search. (d) Test accuracy using posterior predictive distribution vs storage found using group ℓ_1/ℓ_2 with pruning. The optimal student model for this configuration is obtained with group ℓ_1/ℓ_2 pruning. It has approximately 9.9× the number of parameters and 10× the FLOPS of the base student model. Notation: "S" - pareto frontier of the student models, "T" - Teacher, "IS" - Individual Student. The black dashed line denotes the FLOPS/no. of parameters of the base student model having the same architecture as a teacher model.

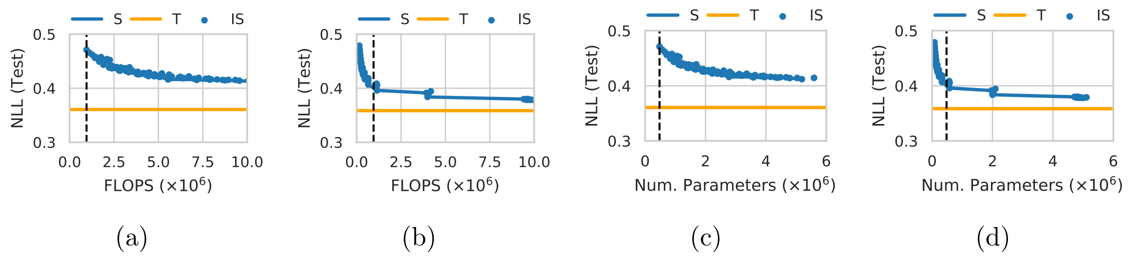


Figure 4.9. NLL-Storage-Computation tradeoff while using Fully-connected networks on MNIST with masking rate 29%. (a) Test negative log likelihood of posterior predictive distribution vs FLOPS found using exhaustive search. (b) Test negative log likelihood of posterior predictive distribution vs FLOPS found using group ℓ_1/ℓ_2 with pruning. (c) Test negative log likelihood of posterior predictive distribution vs storage found using exhaustive search. (d) Test negative log likelihood of posterior predictive distribution vs storage found using group ℓ_1/ℓ_2 with pruning. The optimal student model for this configuration is obtained with group ℓ_1/ℓ_2 pruning. It has approximately 9.9× the number of parameters and 10× the FLOPS of the base student model. Notation: "S" - pareto frontier of the student models, "T" - Teacher, "IS" - Individual Student. The black dashed line denotes the FLOPS/no. of parameters of the base student model having the same architecture as a teacher model.

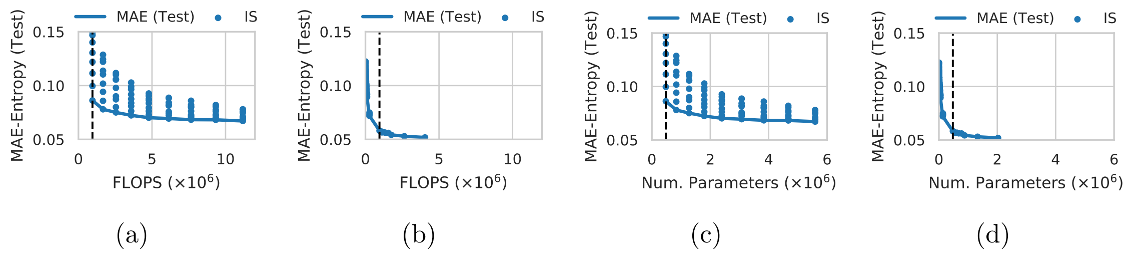


Figure 4.10. Entropy Error-Storage-Computation tradeoff while using Fully-connected networks on MNIST with masking rate 29%. (a) Test mean absolute error for posterior entropy vs FLOPS found using exhaustive search. (b) Test mean absolute error for posterior entropy vs FLOPS found using group ℓ_1/ℓ_2 with pruning. (c) Test mean absolute error for posterior entropy vs storage found using exhaustive search. (d) Test mean absolute error for posterior entropy vs storage found using group ℓ_1/ℓ_2 with pruning. The optimal student model for this configuration is obtained with group ℓ_1/ℓ_2 pruning. It has approximately 4.2× the number of parameters and 4.2× the FLOPS of the base student model. Notation: "MAE (Test)" - pareto frontier of the MAEs obtained using different student models, "IS" - Individual Student. The black dashed line denotes the FLOPS/no. of parameters of the base student model having the same architecture as a teacher model.

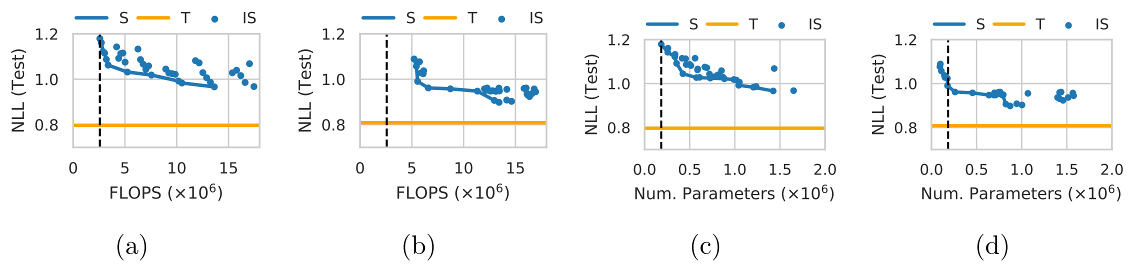


Figure 4.11. NLL-Storage-Computation tradeoff while using CNNs on CIFAR10 with training set size of 20,000 samples. (a) Test negative log likelihood of posterior predictive distribution vs FLOPS found using exhaustive search. (b) Test negative log likelihood of posterior predictive distribution vs FLOPS found using group ℓ_1/ℓ_2 with pruning. (c) Test negative log likelihood of posterior predictive distribution vs storage found using exhaustive search. (d) Test negative log likelihood of posterior predictive distribution vs storage found using group ℓ_1/ℓ_2 with pruning. The optimal student model for this configuration is obtained with group ℓ_1/ℓ_2 pruning. It has approximately 4.7× the number of parameters and 5.2× the FLOPS of the base student model. Notation: "S" - pareto frontier of the student models, "T" - Teacher, "IS" - Individual Student. The black dashed line denotes the FLOPS/no. of parameters of the base student model having the same architecture as a teacher model.

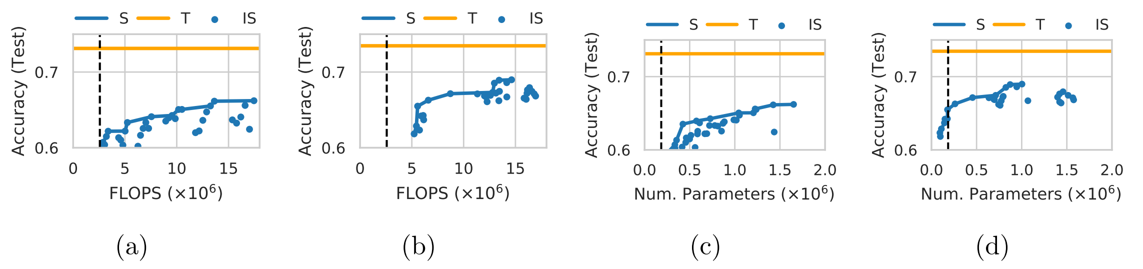
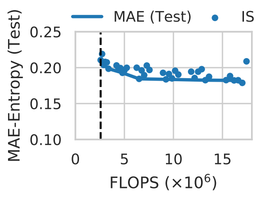
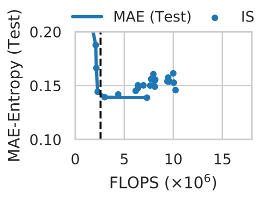


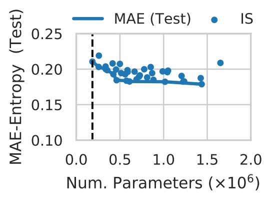
Figure 4.12. Accuracy-Storage-Computation tradeoff while using CNNs on CIFAR10 with sub-sampling training data to 20,000 samples. (a) Test accuracy using posterior predictive distribution vs FLOPS found using exhaustive search. (b) Test accuracy using posterior predictive distribution vs FLOPS found using group ℓ_1/ℓ_2 with pruning. (c) Test accuracy using posterior predictive distribution vs storage found using exhaustive search. (d) Test accuracy using posterior predictive distribution vs storage found using group ℓ_1/ℓ_2 with pruning. The optimal student model for this configuration is obtained with group ℓ_1/ℓ_2 pruning. It has approximately 5.4× the number of parameters and 5.6× the FLOPS of the base student model. Notation: "S" - pareto frontier of the student models, "T" - Teacher, "IS" - Individual Student. The black dashed line denotes the FLOPS/no. of parameters of the base student model having the same architecture as a teacher model.



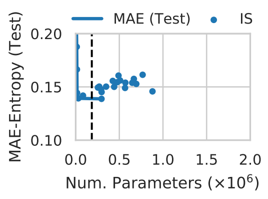
(a)



(b)



(c)



(d)

Figure 4.13. Entropy Error-Storage-Computation tradeoff while using CNNs on CIFAR10 with sub-sampling training data to 20,000 samples. (a) Test mean absolute error for posterior entropy vs FLOPS found using exhaustive search. (b) Test mean absolute error for posterior entropy vs FLOPS found using group ℓ_1/ℓ_2 with pruning. (c) Test mean absolute error for posterior entropy vs storage found using exhaustive search. (d) Test mean absolute error for posterior entropy vs storage found using group ℓ_1/ℓ_2 with pruning. The optimal student model for this configuration is obtained with group ℓ_1/ℓ_2 pruning. It has approximately 1.6× the number of parameters and 2.8× the FLOPS of the base student model. Notation: "MAE (Test)" - pareto frontier of the MAEs obtained using different student models, "IS" - Individual Student. The black dashed line denotes the FLOPS/no. of parameters of the base student model having the same architecture as a teacher model.

4.2.2.4 Experiment 4: Uncertainty Quantification for Downstream Tasks

As noted earlier, uncertainty quantification and decomposition is an important application of Bayesian posterior predictive inference. In this set of experiments, we evaluate our method on two downstream applications: out-of-distribution detection and uncertainty-based ranking. We compare the GPED framework to the full Monte Carlo ensemble as well as to an adaptation of Ensemble Distribution Distillation (EnD^2) [Malinin et al., 2020]. In particular, Malinin et al. [2020] materialize a complete ensemble, which is not feasible in our case due to the large number of samples in the Bayesian ensemble ($\sim 10^5$ samples). We instead use Algorithm 1 with the Dirichlet log likelihood distillation loss used by Malinin et al. [2020]. Additionally, we modify our student models to distill both the predictive distribution and expected data uncertainty in a single model.

Before assessing the performance of these methods on downstream tasks, we first compare their performance in terms of negative log likelihood and MAE on the posterior predictive distribution and expected data uncertainty distillation tasks. We use the same dataset augmentation as in the previous experiment. We compare the GPED and EnD^2 methods using U_o and U_s as well as for small and large model sizes. Note that for distilling entropy under our method in this section, we always use the U_o estimator. Wherever the U_s estimator is mentioned for our method in this section of experiments, it is only applied to distilling predictive means. In Table 4.5 we compare different distillation methods for different model-dataset combinations. These results correspond to the U_s estimator and the largest student model. As an illustration, we present joint and marginal expected data uncertainty distribution plots in Figure 4.15 that correspond to the results in Table 4.5. These figures show how GPED and EnD^2 compare against the Bayesian ensemble on a data case-by-data case basis. Additional results are presented in Tables [4.6- 4.8] and Figure 4.14. The key result of these experiments is that the GPED framework consistently performs better than EnD^2 across all metrics on the test datasets.

Table 4.5. In-distribution Test set metrics comparison using U_s and largest student model obtained using width multiplier.

Model/ Dataset	NLL (Ensemble)	NLL (GPED)	NLL (EnD^2)	MAE Entropy (GPED)	MAE Entropy (EnD^2)
FCNN/ MNIST	0.362	0.408	0.415	0.069	0.105
CNN/ MNIST	0.269	0.296	0.321	0.086	0.106
CNN/ CIFAR10	0.799	0.859	0.907	0.146	0.328

Table 4.6. In-distribution Test set metrics comparison using U_s and base student model matching the teacher architecture.

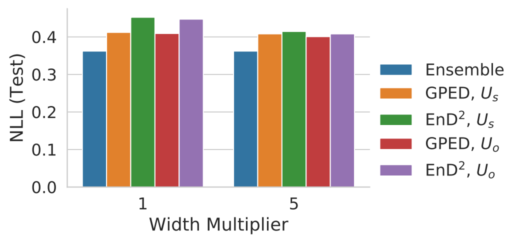
Model/ Dataset	NLL (Ensemble)	NLL (GPED)	NLL (EnD^2)	MAE Entropy (GPED)	MAE Entropy (EnD^2)
FCNN/ MNIST	0.362	0.412	0.452	0.063	0.113
CNN/ MNIST	0.269	0.396	0.460	0.121	0.175
CNN/ CIFAR10	0.799	1.032	1.104	0.181	0.424

Table 4.7. In-distribution Test set metrics comparison using U_o and base student model matching the teacher architecture.

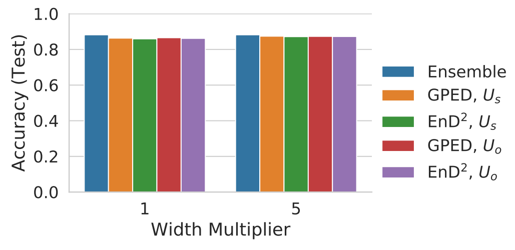
Model/ Dataset	NLL (Ensemble)	NLL (GPED)	NLL (EnD^2)	MAE Entropy (GPED)	MAE Entropy (EnD^2)
FCNN/ MNIST	0.362	0.409	0.447	0.063	0.110
CNN/ MNIST	0.269	0.447	0.460	0.121	0.183
CNN/ CIFAR10	0.799	1.015	1.056	0.181	0.494

Table 4.8. In-distribution Test set metrics comparison using U_o and student model obtained using the largest width multiplier.

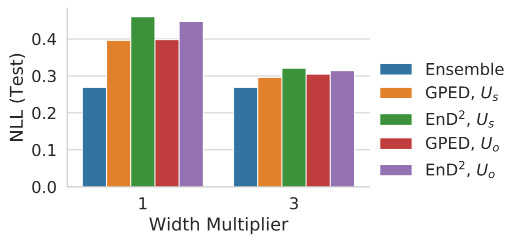
Model/ Dataset	NLL (Ensemble)	NLL (GPED)	NLL (EnD^2)	MAE Entropy (GPED)	MAE Entropy (EnD^2)
FCNN/ MNIST	0.362	0.401	0.408	0.069	0.099
CNN/ MNIST	0.269	0.305	0.314	0.086	0.103
CNN/ CIFAR10	0.799	0.881	0.885	0.146	0.338



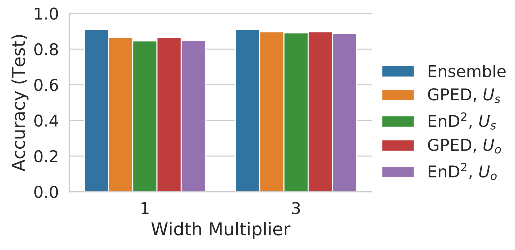
(a) MNIST-FCNN



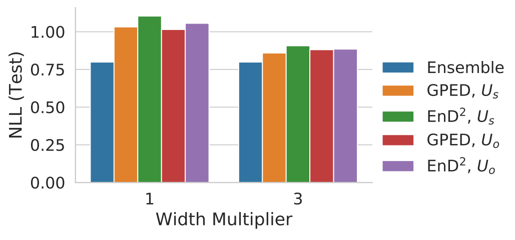
(b) MNIST-FCNN



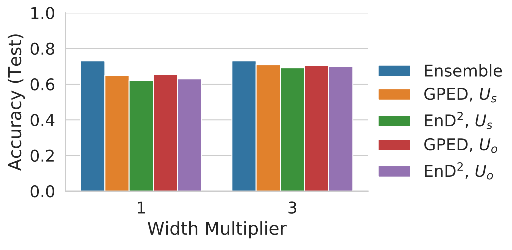
(c) MNIST-CNN



(d) MNIST-CNN

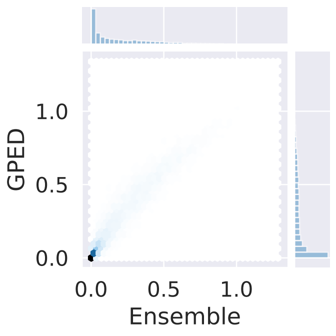


(e) CIFAR10-CNN

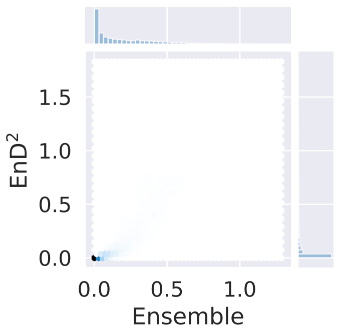


(f) CIFAR10-CNN

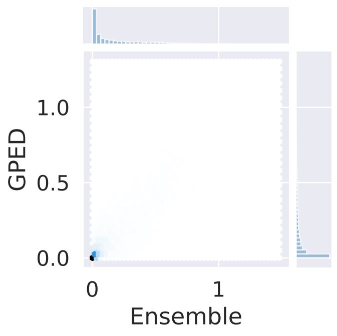
Figure 4.14. Test accuracy and negative log likelihood comparison between GPED and EnD^2 for different dataset-model-estimator combinations.



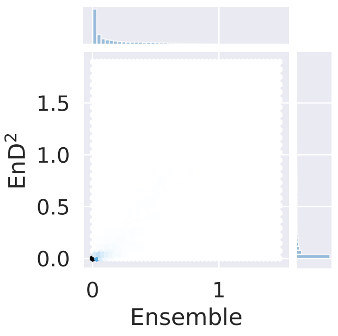
(a) MNIST-FCNN



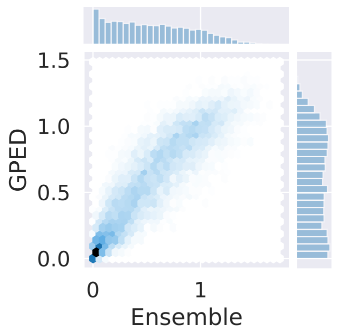
(b) MNIST-FCNN



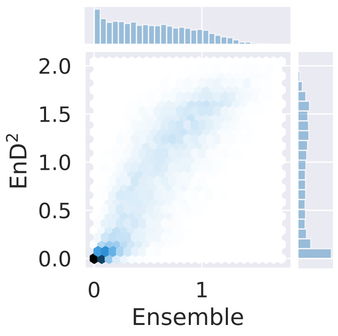
(c) MNIST-CNN



(d) MNIST-CNN



(e) CIFAR10-CNN



(f) CIFAR10-CNN

Figure 4.15. Joint and marginal distributions for expected data uncertainty (also known as the expected entropy) by using U_s estimator for EnD^2 and using the largest model obtained using width multiplier. The expected data entropy is over in-distribution test dataset. An ideal distillation approach would match the marginal distribution of the teacher ensemble given on the top of each plot, as well as have the joint density concentrated on the diagonal. Based on this properties, it is evident that GPED does a better job at tracking the expected data uncertainty.

Out-of-distribution detection: OOD detection has garnered a lot of interest in the deep learning community as it is as a practical challenge during deployment of deep models. In this experiment, we use the measures of total uncertainty and knowledge uncertainty for detecting OOD inputs.

Table 4.9. AUROC for OOD Detection using U_s and largest student model obtained using width multiplier.

Model & Train Data/ OOD Data	Uncertainty	Ensemble	GPED (ours)	EnD^2
FCNN-MNIST/ KMNIST	Total	0.929	0.867	0.816
	Knowledge	0.976	0.928	0.899
FCNN-MNIST/ notMNIST	Total	0.944	0.670	0.652
	Knowledge	0.990	0.762	0.681
CNN-MNIST/ KMNIST	Total	0.894	0.882	0.881
	Knowledge	0.956	0.932	0.952
CNN-MNIST/ notMNIST	Total	0.888	0.882	0.860
	Knowledge	0.946	0.934	0.939
CNN-CIFAR10/ TIM	Total	0.729	0.762	0.721
	Knowledge	0.796	0.808	0.792
CNN-CIFAR10/ LSUN	Total	0.790	0.779	0.747
	Knowledge	0.752	0.767	0.713

OOD detection is a binary classification problem where we utilize a measure of uncertainty to classify an input as in-distribution or out-of-distribution based on a threshold. For our experiments, we use four OOD datasets: KMNIST [Clanuwat et al., 2018a], notMNIST [Bulatov, 2011], TinyImageNet (TIM) (CS231N, 2017), and SVHN [Netzer et al., 2011a]. We run our experiments for different combinations of models, in-distribution datasets, out-of-distribution datasets, model architectures, and estimators used for distilling the predictive distribution under the proposed framework as well as for the EnD^2 framework.

We use the same dataset augmentations as used in Experiment 3. For assessing the performance on downstream tasks with respect to the student model architecture, we use a base student model with the same architecture as the teacher and the largest width multiplier explored in Experiment 3. As noted earlier, we augment the student models under our proposed framework to distill both the predictive distribution and the expected entropy (thus making our network's output dimensionality $C + 1$, where C denotes the number of classes). The output dimensionality of the *prior network* used in the EnD^2 framework is C (for C different concentration parameters) [Malinin et al., 2020]. We use exponential activation at the expected entropy output as well as the prior network output to enforce the positivity constraint. We additionally use a temperature value of $\tau_s = 2.5$ while training all the prior network based models. Malinin et al. [2020] suggest training the prior networks with a temperature annealing schedule, however we find that in our experiments prior networks achieve better performance in terms of log likelihood when using a fixed temperature. The dropout rate for CNN models is set at $p = 0.3$, while FCNN models have a dropout rate of $p = 0.5$. The rest of the experimental details remain the same as stated in Experiment 3.

We report example OOD detection results using the U_s estimator and the largest student model in Table 4.9. Our overall results show that GPED outperforms EnD^2 in 75% of cases across all experimental settings considered (additional results are given in Tables [4.10-4.12]).

Table 4.10. AUROC for OOD Detection using U_s and student model of the same architecture as the teacher.

Model & Train Data/ OOD Data	Uncertainty	Ensemble	GPED (ours)	EnD^2
FCNN-MNIST/ KMNIST	Total	0.929	0.875	0.761
	Knowledge	0.948	0.926	0.861
FCNN-MNIST/ notMNIST	Total	0.944	0.736	0.659
	Knowledge	0.990	0.803	0.720
CNN-MNIST/ KMNIST	Total	0.894	0.829	0.887
	Knowledge	0.956	0.887	0.933
CNN-MNIST/ notMNIST	Total	0.888	0.841	0.828
	Knowledge	0.946	0.902	0.889
CNN-CIFAR10/ TIM	Total	0.729	0.737	0.726
	Knowledge	0.796	0.773	0.773
CNN-CIFAR10/ LSUN	Total	0.790	0.760	0.733
	Knowledge	0.752	0.718	0.653

Table 4.11. AUROC for OOD Detection using U_o and student model of the same architecture as the teacher.

Model & Train Data/ OOD Data	Uncertainty	Ensemble	GPED (ours)	EnD^2
FCNN-MNIST/ KMNIST	Total	0.929	0.847	0.802
	Knowledge	0.948	0.907	0.912
FCNN-MNIST/ notMNIST	Total	0.944	0.782	0.762
	Knowledge	0.990	0.847	0.890
CNN-MNIST/ KMNIST	Total	0.894	0.829	0.884
	Knowledge	0.956	0.890	0.937
CNN-MNIST/ notMNIST	Total	0.888	0.848	0.859
	Knowledge	0.946	0.907	0.918
CNN-CIFAR10/ TIM	Total	0.729	0.747	0.703
	Knowledge	0.796	0.763	0.747
CNN-CIFAR10/ LSUN	Total	0.790	0.759	0.728
	Knowledge	0.752	0.736	0.650

Table 4.12. AUROC for OOD Detection using U_o and student model obtained by largest width multiplier.

Model & Train Data/ OOD Data	Uncertainty	Ensemble	GPED (ours)	EnD^2
FCNN-MNIST/ KMNIST	Total	0.929	0.895	0.833
	Knowledge	0.948	0.944	0.938
FCNN-MNIST/ notMNIST	Total	0.944	0.751	0.714
	Knowledge	0.990	0.826	0.835
CNN-MNIST/ KMNIST	Total	0.894	0.892	0.867
	Knowledge	0.956	0.940	0.956
CNN-MNIST/ notMNIST	Total	0.888	0.887	0.856
	Knowledge	0.946	0.941	0.935
CNN-CIFAR10/ TIM	Total	0.729	0.750	0.705
	Knowledge	0.796	0.798	0.783
CNN-CIFAR10/ LSUN	Total	0.790	0.779	0.745
	Knowledge	0.752	0.753	0.716

Uncertainty-Based Ranking: Another important application of Bayesian neural networks is ranking instances based on uncertainty. Such rankings are used in active learning and other human-in-the-loop decision systems to prioritize uncertain instances for labeling or analysis by human decision makers. This task is sensitive to the correct rank order of in-distribution instances by uncertainty level, whereas the OOD task is only sensitive to the existence of a threshold that separates in and out of distribution instances. To assess how well our distillation framework preserves the relative ranking between the inputs when compared to the full Bayesian ensemble, we compute the Normalized Discounted Cumulative Gain (nDCG) score [Järvelin and Kekäläinen, 2002] for total uncertainty and knowledge uncertainty. A higher nDCG score implies that the correct ranking of inputs is better preserved under the distillation framework. For our experiments, we asses nDCG@20. In Table 4.13, we report the nDCG scores using the U_s estimator and largest student model as example results. Overall, GPED outperforms EnD^2 in 91% of settings considered (additional ranking results are given in Tables [4.14-4.16]).

Table 4.13. nDCG@20 out of 100 randomly selected test inputs using U_s estimator and largest student model . Results reported as mean ± std. dev. over 500 trials.

Model & Data	Uncertainty	GPED (ours)	EnD^2
FCNN-MNIST	Total	0.954 ± 0.02	0.946 ± 0.021
	Knowledge	0.924 ± 0.03	0.941 ± 0.028
CNN-MNIST	Total	0.929 ± 0.034	0.916 ± 0.032
	Knowledge	0.888 ± 0.032	0.876 ± 0.045
CIFAR10	Total	0.935 ± 0.022	0.919 ± 0.027
	Knowledge	0.885 ± 0.033	0.889 ± 0.034

Table 4.14. nDCG@20 out of 100 randomly selected test inputs using U_s estimator and student model matching the architecture of teacher. Results reported as mean ± std. dev. over 500 trials.

Data	Uncertainty	GPED (ours)	EnD^2
FCNN-MNIST	Total	0.947 ± 0.022	0.911 ± 0.031
	Knowledge	0.919 ± 0.030	0.895 ± 0.042
CNN-MNIST	Total	0.852 ± 0.023	0.780 ± 0.040
	Knowledge	0.823 ± 0.038	0.768 ± 0.044
CIFAR10	Total	0.895 ± 0.035	0.853 ± 0.041
	Knowledge	0.825 ± 0.045	0.809 ± 0.051

Table 4.15. nDCG@20 out of 100 randomly selected test inputs using U_0 estimator and student model matching the architecture of teacher. Results reported as mean ± std. dev. over 500 trials.

Data	Uncertainty	GPED (ours)	EnD^2
FCNN-MNIST	Total	0.942 ± 0.023	0.907 ± 0.034
	Knowledge	0.922 ± 0.024	0.907± 0.037
CNN-MNIST	Total	0.871 ± 0.050	0.818 ± 0.048
	Knowledge	0.815 ± 0.050	0.747 ± 0.066
CIFAR10	Total	0.903 ± 0.029	0.840 ± 0.047
	Knowledge	0.854 ± 0.041	0.785 ± 0.082

Table 4.16. nDCG@20 out of 100 randomly selected test inputs using U_0 estimator and student model obtained using the largest width multiplier. Results reported as mean ± std. dev. over 500 trials.

Data	Uncertainty	GPED (ours)	EnD^2
FCNN-MNIST	Total	0.962 ± 0.017	0.940 ± 0.024
	Knowledge	0.953 ± 0.023	0.922 ± 0.020
CNN-MNIST	Total	0.951 ± 0.017	0.908 ± 0.034
	Knowledge	0.920 ± 0.028	0.870 ± 0.036
CIFAR10	Total	0.940 ± 0.021	0.901 ± 0.019
	Knowledge	0.883 ± 0.035	0.883 ± 0.036

4.3 Conclusions

We have presented a framework for distilling expectations with respect to the Bayesian posterior distribution of a deep neural network that significantly generalizes the Bayesian Dark Knowledge approach. Our results show that posterior distillation performance can be highly sensitive to the architecture of the student model, but that architecture search methods can identify student model architectures with improved speed-storage-accuracy trade-offs. We have also demonstrated that the proposed approach performs well on downstream tasks that leverage entropy distillation for uncertainty decomposition.

There are many directions for future work, including considering the distillation of a broader class of posterior statistics, and developing more advanced architecture search methods. It is also worth exploring the fidelity of the student models to the teacher models, as it can give us insights into where the student model significantly differs from the teacher ensemble [Stanton et al., 2021].

CHAPTER 5

IMPACT OF PARAMETER SPARSITY ON STOCHASTIC GRADIENT MCMC METHODS FOR BAYESIAN DEEP LEARNING

Bayesian methods hold significant promise for improving the uncertainty quantification ability and robustness of deep neural network models. Recent research has seen the investigation of a number of approximate Bayesian inference methods for deep neural networks, building on both the variational Bayesian and Markov chain Monte Carlo (MCMC) frameworks. A fundamental issue with MCMC methods is that the improvements they enable are obtained at the expense of increased computation time and model storage costs. In this chapter, we investigate the potential of sparse network structures to flexibly trade-off model storage costs and inference run time against predictive performance and uncertainty quantification ability. We use stochastic gradient MCMC methods as the core Bayesian inference method and consider a variety of approaches for selecting sparse network structures. Surprisingly, our results show that certain classes of randomly selected substructures can perform as well as substructures derived from state-of-the-art iterative pruning methods while drastically reducing model training times.

The rest of the chapter is structured as follows. In Section 5.1, we describe the different pruning strategies that we use along with the SGHMC algorithm. Next, in Section 5.2 we describe our experiments and present the results. Finally, in Section 5.3 we present the conclusion of our findings and highlight some potential future directions.

5.1 Methods

In this section, we describe the methods that we apply to study the properties of stochastic gradient MCMC applied to substructures of sparse deep neural networks. We consider a two-step process. Given an initial model architecture, we first apply a method to select a sparse substructure from the initial dense model. We select sparse substructures at the connection level using weight-level sparsity masks. We then study the properties of Bayesian inference applied within the selected substructures. In this section, we first describe methods for selecting sparse network structures. We then provide details for the stochastic gradient MCMC method that we apply.

5.1.1 Selecting Sparse Substructures

We let $\theta_l \in \mathbb{R}^{K_l}$ be the parameter vector for layer l and K_l be the number of parameters in layer l. We define $\mathbf{m} \in \{0,1\}^K$ to be a weight-level sparsity mask and $\mathbf{m}_l \in \{0,1\}^{K_l}$ be the sparsity mask for layer l. For defining parameters and weight-level sparsity masks at iteration i, we use the notation θ^i and $\mathbf{m}^i$ respectively. Given a sparsity mask $\mathbf{m}$ and a parameter vector θ, the masked parameters are given by the Hadamard product $\tilde{\theta} = \mathbf{m} \odot \theta$. Below we briefly describe several approaches to selecting sparse sub-network structures. We emphasize that the contribution of this work is not to propose new methods for deriving sparse sub-networks, but rather to evaluate the impact of sparse sub-network structures on MCMC-based inference methods, a question that has not been addressed in prior research.

5.1.1.1 Iterative Pruning (IP)

The first approach that we consider for selecting a sparsity mask is based on the iterative pruning method of Han et al. [2015]. The iterative pruning approach can use any standard gradient-based optimization method as the base learning method. The algorithm begins with a randomly selected parameter vector $\theta^0 \in \mathbb{R}^K$ with all weights active ($\mathbf{m}^0 = 1$). The algorithm proceeds over T pruning iterations. In each iteration

i, a specified number of epochs ϵ of the base learning method are applied yielding a final parameter vector θ^i for that iteration. At the end of the iteration, a fixed fraction π of the active weights are pruned resulting in a new sparsity mask $\mathbf{m}^{i+1}$. The active weights with the smallest magnitudes are selected for pruning on each iteration. The initial weight vector for iteration $i+1$ is then given by $\theta^i \odot \mathbf{m}^{i+1}$. This method greedily produces a nested sequence of increasingly sparser network substructures encoded by the sparsity masks $\mathbf{m}^i$ along with corresponding locally optimal parameters θ^i.

5.1.1.2 Iterative Pruning with Rewinding (IPR)

Frankle and Carbin [2018] proposed an iterative pruning method that is very similar to the method of Han et al. [2015], which we will refer to as Iterative Pruning with Rewinding. This approach differs from basic iterative pruning in that following each pruning iteration, the weights used to initialize the next iteration of the algorithm are formed by combining the updated mask $\mathbf{m}^{i+1}$ with the original random weight vector θ^0 instead of the weight vector obtained at the end of iteration i. Effectively, the active weights are rewound to their initial values at the start of each round of iterative pruning. We will again define θ^i to be the value of the weights at the end of iteration i. This method also greedily produces a nested sequence of increasingly sparse network structures encoded by the sparsity masks $\mathbf{m}^i$ along with corresponding locally optimal point-estimated parameters θ^i. However, as Frankle and Carbin [2018] show, the pairs $(\theta^0, \mathbf{m}^i)$ appear to have somewhat special properties with respect to optimization-based learnability. We investigate whether the properties of the sub-structures and initial parameters identified by iterative pruning with rewinding differ from those identified by iterative pruning when sparse sub-structures are used in the context of MCMC-based Bayesian deep learning.

5.1.1.3 Random Layer-wise Masking (RLM)

A question that is unexplored in the both Han et al. [2015] and Frankle and Carbin [2018] from the optimization perspective is the importance of the exact sparse structure obtained by iterative pruning relative to the sparsity level per network layer. We study a Layer-wise Random Masking approach to answer this question in the case of Bayesian deep learning. This approach requires the specification of a desired sparsity rate π_l for each layer l of the network. Given the desired sparsity rate π_l, we uniformly sample a random sparsity mask $\mathbf{m}_l$ for layer l from the space of all binary vectors with this sparsity rate. These layer-wise random masks are then assembled into a sparsity mask $\mathbf{m}$ for the complete network. We explore three approaches to select the desired sparsity levels π_l. The first approach runs iterative pruning for T iterations, obtains the final sparsity mask, and computes the corresponding sparsity rate π_l for each layer. We denote this approach by RLM(IP). Iterative pruning with rewinding can be used in the same way, obtaining an approach we denote by RLM(IPR). These two approaches allow us to compare the optimized sparse network structures obtained using iterative pruning to networks where the sparsity rates per layer have been optimized while the specific sparsity structures are randomly selected. The final variant of the random layer-wise approach simply sets the same fixed sparsity rate per level $\pi_l = \pi$. This approach allows us to investigate how critical preserving the per-layer sparsity rates are. We refer to this approach as RLM(F).

5.1.1.4 Random Global Masking (RGM)

Random global making is the most basic mask generation approach that we consider. We specify a fixed sparsity rate π and select a random mask $\mathbf{m} \in \{0, 1\}^K$ from the space of all binary vectors yielding this sparsity level. This approach allows us to study how critical the distribution of sparsity over network levels is compared to the overall level of sparsity. As we conclude this subsection, we emphasize that RLM(F) and

RGM do not require us to run any form of iterative pruning, resulting in significant training time and resource savings when compared to the other described methods.

5.1.2 SGHMC in Sparse Structures

Given a sparse neural network structure represented by a weight-level sparsity mask $\mathbf{m}$, we perform stochastic gradient Hamiltonian Monte Carlo (SGHMC) within the specified substructure. SGHMC only requires the computability of an unnormalized version of the log parameter posterior density: $U(\theta) = -\log p(\theta|\lambda) - \sum_{i=1}^{N} p(y_i|\mathbf{x}_i, \theta)$. Like all HMC methods, SGHMC is based on sampling in an extended space that includes the model parameters $\theta \in \mathbb{R}^K$ as position variables as well as an auxiliary set of momentum variables $\mathbf{r} \in \mathbb{R}^K$. This sampling process simulates discretized Hamiltonian dynamics. SGHMC methods compute a stochastic approximation of the gradient $\nabla U(\theta)$ of the unnormalized energy function $U(\theta)$ using a mini-batch of data $\mathcal{D}'_{tr} \subset \mathcal{D}_{tr}$ of size B. When performing Bayesian inference over sparse substructures, the stochastic gradient $\nabla\tilde{U}(\theta)$ only needs to be computed over the selected substructure, and doing so has the potential to significantly accelerate inference time. For simplicity of implementation, we use a less computationally efficient masking approach during training that allows existing neural network model implementations to be used.

The complete SGHMC algorithm that we use for sampling within substructures is defined in Algorithm 2. In this algorithm, $\theta^{(0)}$ is an initial parameter vector with sparsity structure matching that specified by the sparsity mask $\mathbf{m}$. α_s is the step size used on sampling iteration s. η is a friction term introduced to counteract the additional noise introduced by using stochastic gradients. S is the number of desired sampled. Algorithm 2 adds a masking step to the basic SGHMC algorithm used by Zhang et al. [2020]. The masking step projects sampled parameters back into the required substructure on each sampling iteration. This algorithm correctly samples from the posterior distribution of the weights in the selected substructure due to the fact that the updates for both $\mathbf{r}$ and $\theta^{(s)}$ do not mix information across their dimensions. For large models, running SGHMC from a randomly chosen $\theta^{(0)}$ can result in the need for very long burn-in times. To help mitigate this issue, we always initialize $\theta^{(0)}$ using the output of an SGD-based optimization algorithm applied within the substructure corresponding to $\mathbf{m}$. This optimization step itself requires an initial setting of the model parameters. We initialize at random for random masks. For iterative pruning-based methods, we use the final parameter vector returned along with the sparsity mask.

Algorithm 2 Sparse Sub-Structure SGHMC

```
  **Inputs:** $\theta^{(0)}$, $\mathbf{m}$, $\mathcal{D}_{tr}$, $B$, $\alpha_{0:S}$, $\eta$, $S$
  $\mathbf{r} \leftarrow 0$
  **for** $(0 \leq s \leq S)$ **do**
      Select mini-batch $\mathcal{D}'_{tr} \subset \mathcal{D}_{tr}$ of size $B$
      $\nabla\hat{U} \leftarrow -\nabla p(\theta^{(s)}|\lambda) - \frac{N}{B}\sum_{(\mathbf{x},y)\in\mathcal{D}'} \nabla p(y|\mathbf{x}, \theta^{(s)})$
      $\mathbf{r} \leftarrow (1-\eta)\mathbf{r} - \alpha_k \nabla\hat{U} + \epsilon \cdot \sqrt{2\eta\alpha_k}, \quad \epsilon \sim \mathcal{N}(0, I)$
      $\theta^{(s+1)} \leftarrow \mathbf{m} \odot (\theta^{(s)} + \mathbf{r})$
  **end for**
  **Return:** $[\theta^{(1)}, ..., \theta^{(S)}]$
```

5.2 Experiments and Results

In this section, we present experiments and results. Each of the experiments involves the composition of several components including a base neural network architecture, a data set, a method for selecting a sparse substructure (see Section 5.1.1), and an optimization or Bayesian inference approach (see Section 5.1.2). We focus on classification as a task and use three model/data set combinations. We describe these combinations below.

5.2.1 Experimental Protocols

To evaluate the effect of sparsity on Bayesian inference in fully connected architectures, we use the Fashion MNIST data set (FMNIST) [Xiao et al., 2017] with a two-hidden layer MLP with 200 hidden units per hidden layer (MLP200). The FMNIST data set has 10 classes, 60K training instances and 10K test instances. For a larger-scale model, we pair the 20-layer residual network model (ResNet20) used in Frankle and Carbin [2018] with the CIFAR10 data set [Krizhevsky et al., 2009]. CIFAR10 has 10 classes, 50K training instances and 10K test instances. For a more challenging data set, we pair the ResNet20 model with the CIFAR100 data set [Krizhevsky et al., 2009]. CIFAR100 has 100 classes, 50K training instances and 10K test instances. All data sets are freely available. Additionally, we use an isotropic Gaussian prior for all models.

For evaluating performance on the classification task, we use the metrics of accuracy ($\uparrow$), negative log likelihood (NLL, $\downarrow$), and expected calibration error (ECE, $\downarrow$). For assessing the efficiency of MCMC samplers, we use the metrics of effective sample size (ESS, $\uparrow$), and auto-correlation coefficient (ACF $\downarrow$). We note the definitions of ESS and ASF below.

Autocorrelation Function (ACF): Autocorrelation functions are used to assess how correlated a univariate sequence with itself. The ACF is a useful tool for analyzing the correlation between functions of samples collected via a Markov chain. We select the instantaneous negative log likelihood function on the test set $\text{iNLL}^{(s)} = -(1/N_{te})\sum_{(\mathbf{x},y)\in\mathcal{D}_{te}} \log p(y|\mathbf{x}, \theta^{(s)})$ as the summary statistic as it is a convenient way to summarize high dimensional sampled parameters. If nearby samples are very similar at a given lag, then the instantaneous log likelihood values will also be similar and the ACF will be high. Samples that are approximately IID would have ACF near zero at all lags.

Effective Sample Size (ESS): The effective sample size is used to assess the effective number of independently drawn samples materialized from a collection of Markov chains [Robert and Casella, 2013, Page 499]. The ESS takes into account both the variance within and between chains, and therefore summarizes the correlation between samples. High values indicate a more efficient Markov chain, where the samples are closer to being independent. As an example, completely independent samples would correspond to an ESS equal to the total number of samples collected. We use the implementation provided in Pyro to compute the ESS [Bingham et al., 2019].

For hyperparameter optimization, we start with the hyperparameters from Frankle and Carbin [2018] and further tuned the learning rate and scheduler to improve on their performance. We found the following hyperparameters to perform better.

- **MLP200/FMNIST:**
  - **SGD:** For all use-cases, whether for pre-training or for IP/IPR, we use standard SGD with a learning rate of 0.01, a weight decay of 1×10^{-3}, a momentum of 0.9, and 60 epochs. For the learning rate scheduler, we use a multistep scheduler, which reduces the learning rate by a factor of 0.1 at epochs 20 and 40.
  - **SGHMC:** For all implementations of SGHMC, we initialize the weights using SGD (as described above). We allow for a burn in of 50 epochs and then collect the number of pre-specified samples (e.g. for one chain we collect 50 samples). The learning rate is set to a constant value of $\alpha_s = 0.01$, the friction term $\eta = 0.1$ (i.e. momentum is 0.9), the prior precision is set to 60 ($1 \times 10^{-3} \times |\mathcal{D}_{tr}|$)
- **ResNet20/CIFAR10:**
  - **SGD:** We use SGD with a learning rate of 0.01, a weight decay of 1×10^{-4}, a momentum of 0.9, and 160 epochs. For the learning rate scheduler, we use a multistep scheduler, which reduces the learning rate by a factor of 0.1 at epochs 80 and 120.

  - **SGHMC:** As for the MLP200, we initialize the weights using SGD (as described above). We allow for a burn in of 50 epochs and then collect the number of pre-specified samples. The learning rate follows the cosine annealing scheduler, with an initial learning rate of 0.2 and a final value of 0.0. The friction term $\eta = 0.5$ (i.e. momentum is 0.5), the prior precision is set to 60 ($1 \times 10^{-4} \times |\mathcal{D}_{tr}|$)

- **ResNet20/CIFAR100:**
  - **SGD:** We use the same settings as for ResNet20/CIFAR10.
  - **SGHMC:** We use the same settings as for ResNet20/CIFAR10.

For this chapter, we use the SGHMC implementation provided by Vadera et al. [2020a]. We also utilize parts of the PyTorch implementation provided by Frankle and Carbin [2018] for our experiments.

5.2.2 Experiment 1: How does SGHMC perform when applied to optimized sparse substructures?

In this experiment, we evaluate the performance of SGHMC applied to substructures selected using the iterative pruning (IP-SGHMC) and iterative pruning with re-winding (IPR-SGHMC) methods. We compare to the baseline of SGHMC applied to the full, dense model (Full-SGHMC). The results are presented in Figure 5.1. As we can see, the drop in accuracy at 83% sparsity is less than 3% on FMNIST and CIFAR10 for both IP-SGHMC an IPR-SGHMC. On CIFAR100, IP-SGHMC performs well at 83% sparsity while IPR-SGHMC shows a larger drop. As expected, both methods show decreased accuracy as sparsity increases on all data sets. However, the drop in accuracy is minimal (less than 4%) on both FMNIST and CIFAR10 even at 95% sparsity. The degradation in accuracy is more pronounced on the more challenging CIFAR100 data set. We can also see that the negative log likelihood increases for both methods on both data sets as a function of sparsity. However, there is no clear advantage for either method. Finally, we can see that the expected calibration errors are very low on both FMNIST and CIFAR10 for both methods in terms of absolute level, while on CIFAR100 IP-SGHMC appears to exhibit poor calibration at lower sparsity levels.

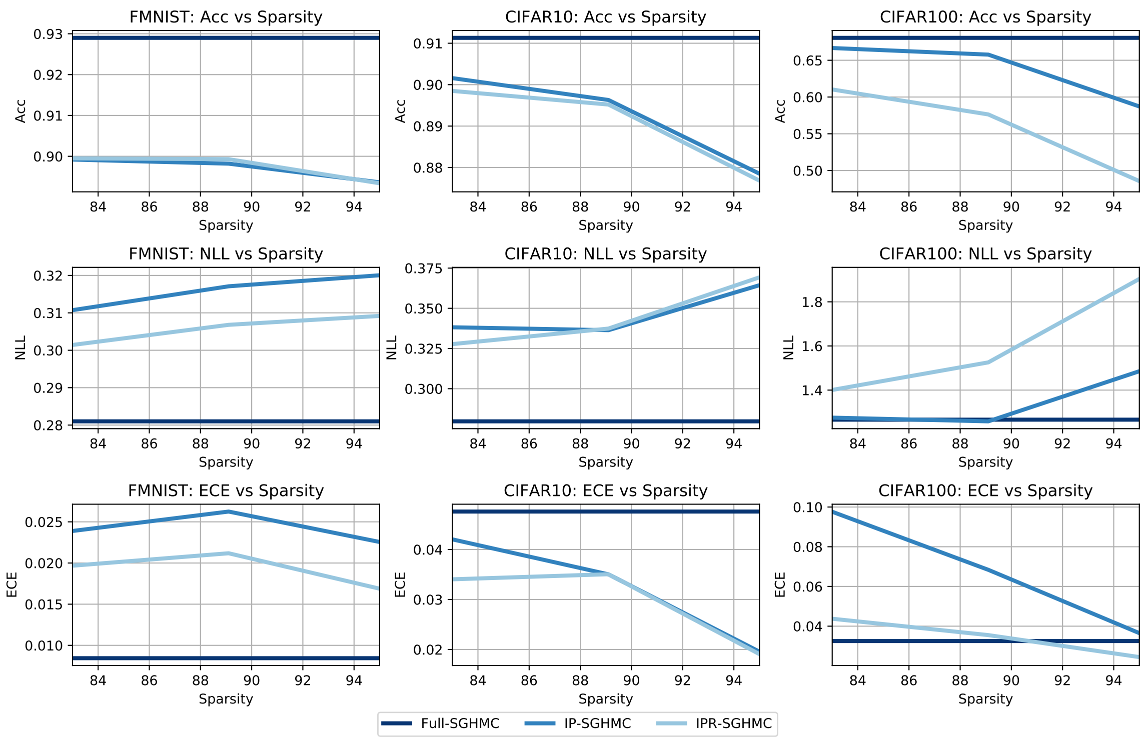


Figure 5.1. Performance of SGHMC applied to optimized sparse substructures compared to SGHMC applied to full models.

To provide context for the trade-off between storage and performance, in Table 5.1 we show the total number of parameters that require storage under Full-SGHMC and IP-SGHMC on FMNIST and CIFAR10. The parameter counts for IPR-SGHMC are identical to those of IP-SGHMC and the parameters required for CIFAR100 are nearly identical to those required for CIFAR10 for all methods. We also include the parameter count for the base models (Full-OPT) for reference. Importantly, for the SGHMC methods, we report the total number of parameters in the complete sparse

Table 5.1. Total no. of parameters by sparsity rate (↓) and wall-clock time per image during testing (↓).

Dataset	Method	Sparsity	# Params
FMNIST	Full-SGHMC	0%	9960500
FMNIST	IP-SGHMC	83%	1695300
FMNIST	IP-SGHMC	89%	1085700
FMNIST	IP-SGHMC	95%	468150
FMNIST	Full-OPT	0%	199210
CIFAR10	Full-SGHMC	0%	13623700
CIFAR10	IP-SGHMC	83%	2318800
CIFAR10	IP-SGHMC	89%	1485000
CIFAR10	IP-SGHMC	95%	640350
CIFAR10	Full-OPT	0%	272474

ensemble. As we can see, even at the lowest level of sparsity tested (83%), the savings in storage for the sparse posterior ensembles is highly significant compared to the dense posterior ensembles. At 95% sparsity, the sparse posterior ensembles have roughly twice the storage cost of the base model (Full-OPT). The trade-off between storage and performance appears to be quite favorable for both FMNIST and CIFAR10 at the lower sparsity levels relative to the full posterior ensemble.

Table 5.2. Time comparison of inference on a CPU.

Dataset	Method	Sparsity	Inference Time/ data point (s)	Inference speedup	Hardware
FMNIST	Full-SGHMC	0%	0.0104	1×	Apple Macbook Pro (8 GB memory, M1)
FMNIST	IP-SGHMC	83%	0.0097	1.03×	
FMNIST	IP-SGHMC	89%	0.0067	1.55×	
FMNIST	IP-SGHMC	95%	0.0032	3.25×	
FMNIST	IP-SGHMC	99.5%	0.0006	17.33×	
FMNIST	Full-OPT	0%	0.0002	50×	
FMNIST	Full-SGHMC	0%	0.0139	1×	Google Colab (13 GB memory, Intel Xeon@2.20GHz)
FMNIST	IP-SGHMC	83%	0.0104	1.33×	
FMNIST	IP-SGHMC	89%	0.0069	2×	
FMNIST	IP-SGHMC	95%	0.0031	4.48×	
FMNIST	IP-SGHMC	99.5%	0.0007	19.86×	
FMNIST	Full-OPT	0%	0.0003	50×	

We also provide a prediction inference time speedup comparison relative to the dense ensemble (Full-SGHMC) for the MLP200/FMNIST model in Table 5.2. For computing the speedups, we implemented the MLP200 model using the GNU Scientific Library's (GSL) sparse matrix multiplication functions [Kernighan and Ritchie, 2006, Galassi et al., 2003] and ran experiments on an Apple MacBook Pro (8 GB Memory, M1 processor). As we can see, at 89% sparsity we achieve an $\approx 1.6\times$ speedup, while at 95% sparsity we achieve $\approx 16\times$ speedup. These are highly significant improvements in prediction inference time relative to the modest drop in prediction performance on FMNIST.[1]

5.2.3 Experiment 2: How does SGHMC within random sparse substructures compare to SGHMC within optimized sparse substructures?

While prior work on optimization-based learning indicates that directly learning high-quality models in compact or sparse models works less well than iterative pruning methods, iterative pruning methods are extremely slow to execute due to the large number of epochs needed to achieve high rates of sparsity. In this experiment we investigate the performance of different approaches to random generation of sparse substructures including random layer-wise masking with fixed sparsity rates per level (RLM(F)), as well as random global masking (RGM). We compare these against Full-SGHMC and IPR-SGHMC. We also compare to a random layer-wise masking variant that uses the per-layer sparsity levels obtained from iterative pruning with rewinding (RLM(IPR)). This approaches matches the per-layer sparsity level of the substructure identified by iterative pruning while randomizing the actual substructure. Since producing random substructures requires drastically reduced computation compared to iterative pruning, we sample several random substructures and run separate chains within each.

[1]We note that we do not include speedup results for convolutional models, but prior research shows these speedups to be in the range of $3-7\times$ as noted previously Park et al. [2017], Li et al. [2017].

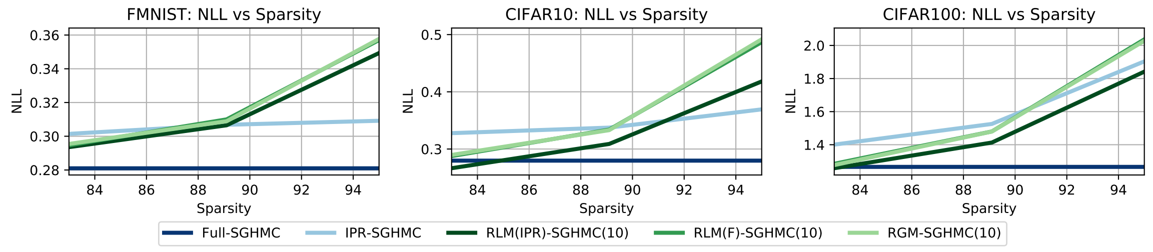


Figure 5.2. Performance of SGHMC applied in random sparse substructures compared to SGHMC applied to full models.

Table 5.3. Total training time (↓) for CIFAR10.

Method	Chains	Sparsity	Time (s)
IPR-SGHMC	1	83%	19,244
IPR-SGHMC	1	89%	22,866
IPR-SGHMC	1	95%	30,111
RLM(F)/RGM-SGHMC	1	-	2,943
RLM(F)/RGM-SGHMC	5	-	12,452
RLM(F)/RGM-SGHMC	10	-	24,338

In Figure 5.2 we show the negative log likelihood results based on randomly sampling 10 substructures and running 10 separate chains. We divide the total sample budget evenly among the 10 chains. Surprisingly, the use of random substructures and parallel chains performs as well as or better than the use of a single SGHMC chain with a sparse sub-structure using iterative pruning at lower levels of sparsity. At higher levels of sparsity, iterative pruning outperforms random substructures with parallel chains. We also see that matching the per-layer sparsity levels of iterative pruning does produce improvements over uniform per-layer sampling.

In Table 5.3, we show the total training time for several of the methods investigated in this experiment. As we can see, the time required by the IPR-SGHMC method is very high due to the need to run iterative pruning to derive the sparsity mask before sampling can run. The training time also increases with the desired level of sparsity. Due to the fact that our masking implementation of SGHMC does not realize a speedup due to sparsity when sampling, the time required to run the random layer-wise masking approach with fixed sparsity per level (RLM(F)-SGHMC) and the random global masking approach (RGM-SGHMC) are the same and independent of the sparsity level (including sampling in the fully dense model). As we can see, running 10 SGHMC chains at any level of sparsity requires total time approximately equal to running iterative pruning followed by one chain at the 89% sparsity level. However, the 10 RLM(F)-SGHMC chains can be run completely in parallel within an actual wall-clock period equal to the total time to run one chain. Overall, these results indicate that the use of iterative pruning appears to be unnecessary at lower sparsity levels so long as multiple chains are used to offset the selection of random substructures. It also suggests a number of other interesting possibilities such as running iterative pruning to further sparsify a randomly selected sparse structure, which we leave to future work. Additional results from this experiment are presented in Figures 5.3-5.5.

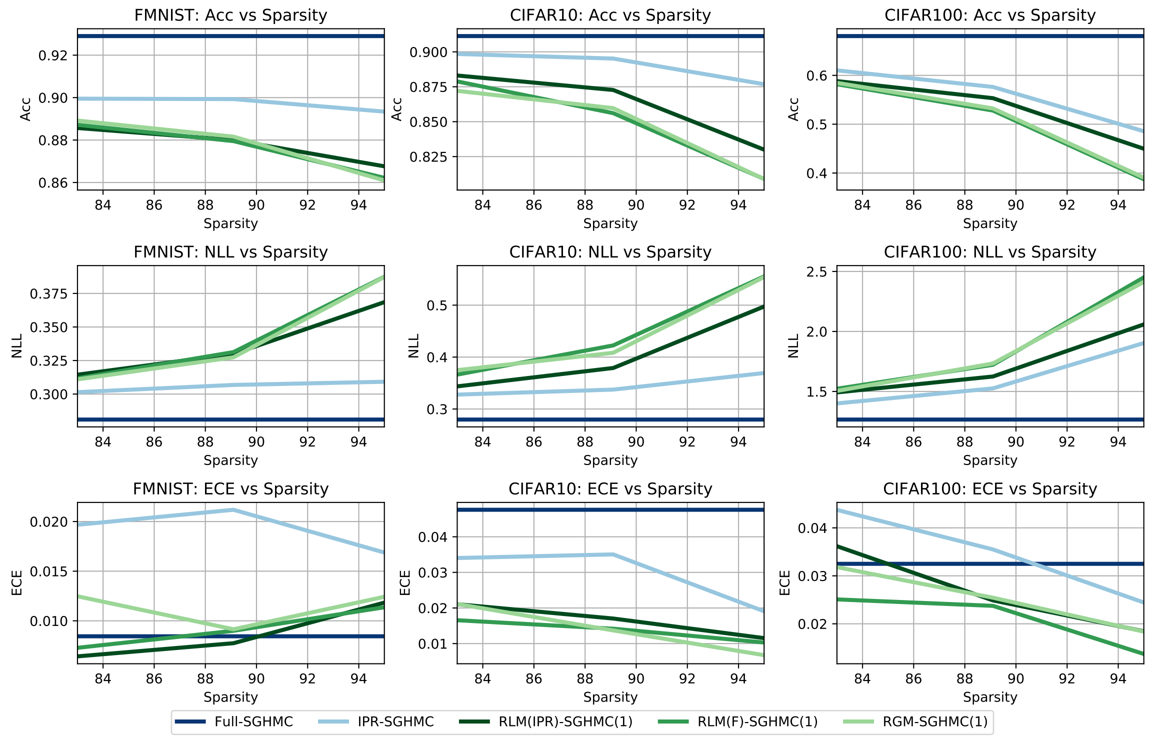


Figure 5.3. Performance comparison of SGHMC applied in random sparse sub-structure compared to SGHMC applied to full models on the FMNIST, CIFAR10 and CIFAR100 datasets. In this figure, we sample 1 random sparse sub-structure using each for the random sparse sub-structure selection methods denoted in the legend, and apply SGHMC to it. We collect total 50 samples from the chain.

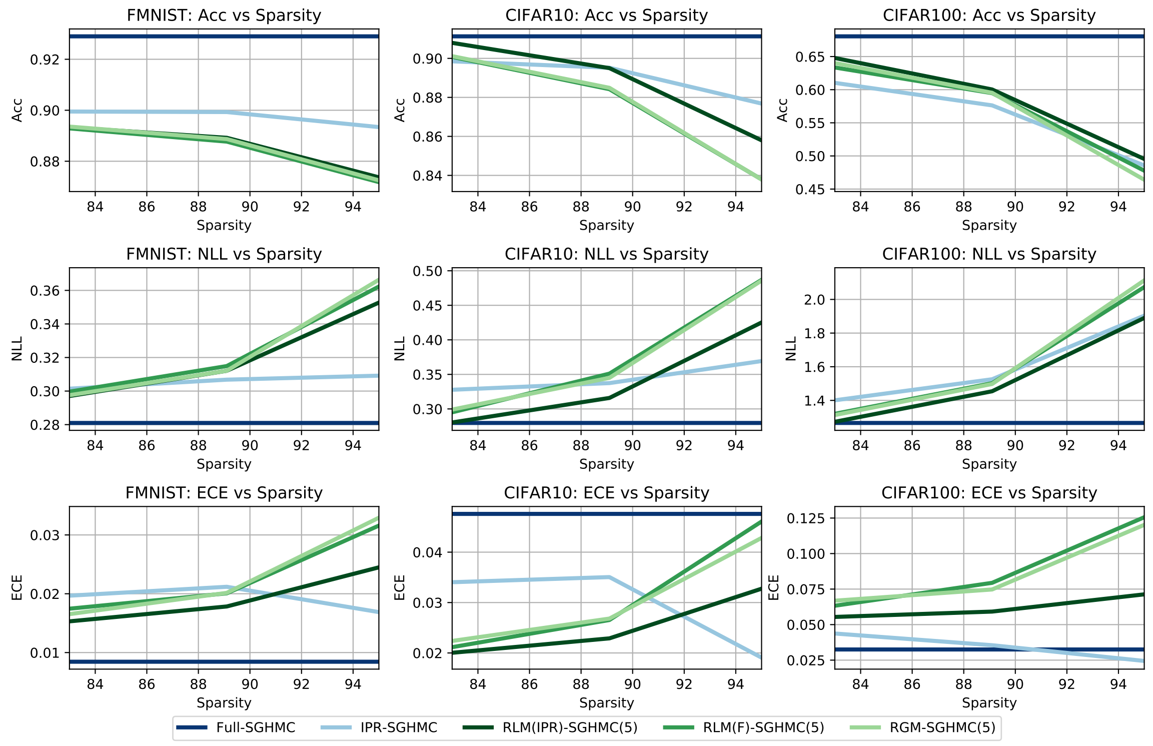


Figure 5.4. Performance comparison of SGHMC applied in random sparse sub-structure compared to SGHMC applied to full models on the FMNIST, CIFAR10 and CIFAR100 datasets. In this figure, we sample 5 random sparse sub-structures each using the random sparse sub-structure selection methods denoted in the legend, and run 5 parallel SGHMC chains. We collect 10 samples from each chain to obtain a total of 50 samples across 5 chains.

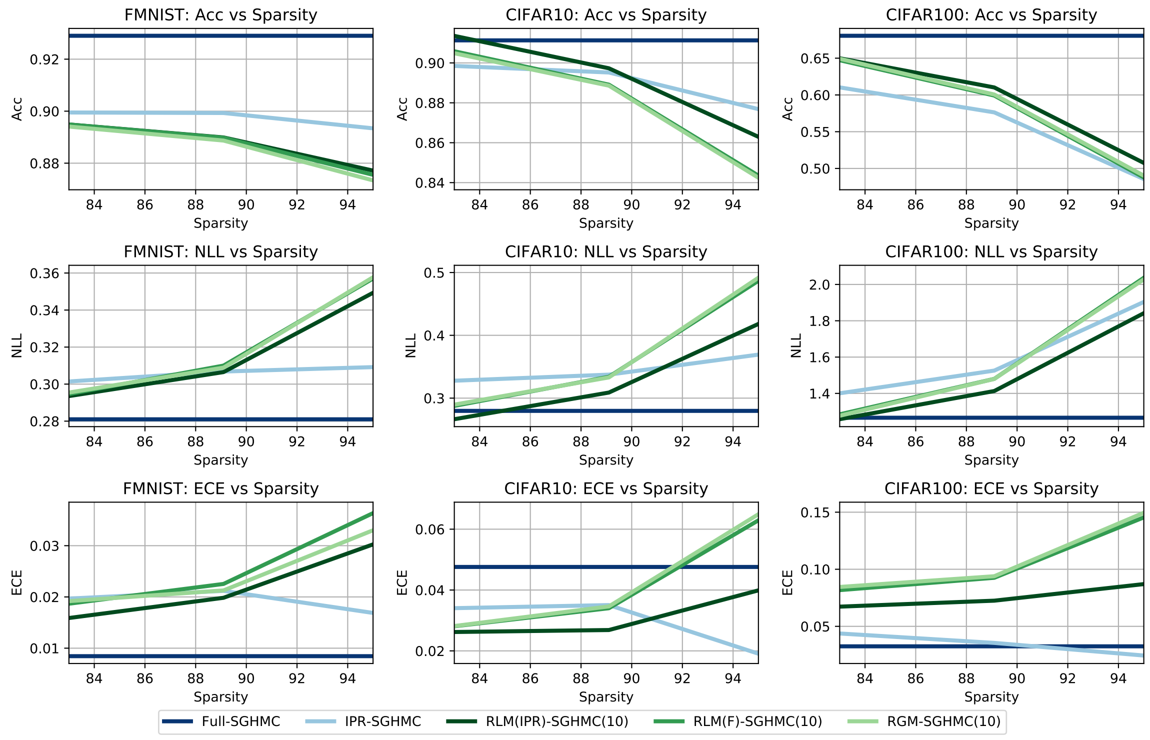


Figure 5.5. Performance comparison of SGHMC applied in random sparse sub-structure compared to SGHMC applied to full models on the FMNIST, CIFAR10 and CIFAR100 datasets. In this figure, we sample 10 random sparse sub-structures each using the random sparse sub-structure selection methods denoted in the legend, and run 10 parallel SGHMC chains. We collect 5 samples from each chain to obtain a total of 50 samples across 10 chains.

Table 5.4. Mean ESS (↑) on CIFAR10.

Sparsity	0 %	89%	95%
IPR-SGHMC	39.8224	41.5218	42.0768
RLM(IPR)-SGHMC	-	38.5486	38.6837
RGM-SGHMC	-	37.7584	37.9131

5.2.4 Experiment 3: How does sampling within different types of sparse structures affect the underlying SGHMC Markov chain?

In this experiment, we examine the structure of the SGHMC Markov chains that result from the use of different types of sparse structures. Prior observations regarding the difficulty of optimizing compact and sparse models suggests that mixing in a gradient-based sampler such as SGHMC might degrade in sparse models. However, the size of the space that the sampler operates in is also drastically reduced at high sparsity rates, which we might expect to improve mixing. The overall effect of parameter sparsity on MCMC methods has not been previously investigated.

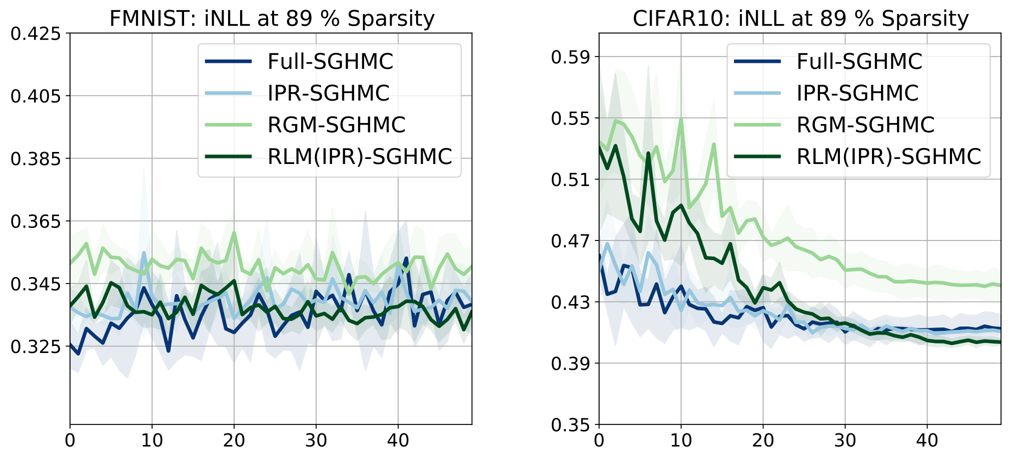


Figure 5.6. Plots of iNLL for FMNIST and CIFAR10 corresponding to sparsity level 89%. We also include the full model as a baseline.

Figure 5.6 shows the instantaneous negative log likelihood (iNLL) on the test set for several methods applied to the FMNIST and CIFAR10 data sets. Based on the iNLL plot, the burn-in time does appear to vary by the method on CIFAR10 with random sub-substructures requiring longer to burn in than optimized substructures and the full model. This observation may be due to the use of an optimization method to select the starting state of each chain, with the iterative pruning optimization process resulting in better initial states than the application of standard optimization methods to random sparse structures. However, all chains appear to be approximately stationary by 50 sampling epochs. On FMNIST, there appears to be much less sensitivity with respect to substructures and all chains burn in quickly. To assess mixing of the MCMC chain, we compute the ESS for each of the posterior ensembles. The mean ESS values are presented in Table 5.4. The results presented in Table 5.4 suggest no significant difference in ESS values as we increase the sparsity. Furthermore, as we move away from IPR based methods to random methods, we again observe no significant change in ESS values. Thus, while there is no significant improvement in ESS values when moving from dense ensembles to sparse ensembles, there's also no decrease in the ESS values. This is promising as it shows that mixing in sparse subspaces is no worse than the original space. Additionally, we also provide ACF plots for the iNLL from the 51st epoch to the 100th epoch (i.e. post burn-in) for each chain. These plots, along with the full set of iNLL plots are presented in Figures 5.7 - 5.9).

5.2.5 Experiment 4: How does SGHMC compare to optimization-based learning of sparse substructures?

In this experiment, we compare full models using both optimization-based learning and SGHMC to sparse models learned using both optimization and SGHMC. We also compare to randomly-selected substructures. The results are presented in Figure 5.10. We again focus on iterative pruning with rewinding as the optimization-based method for deriving sparse structures. We focus on negative log likelihood as the performance measure. First, as we would expect, SGHMC out-performs optimization-based learning

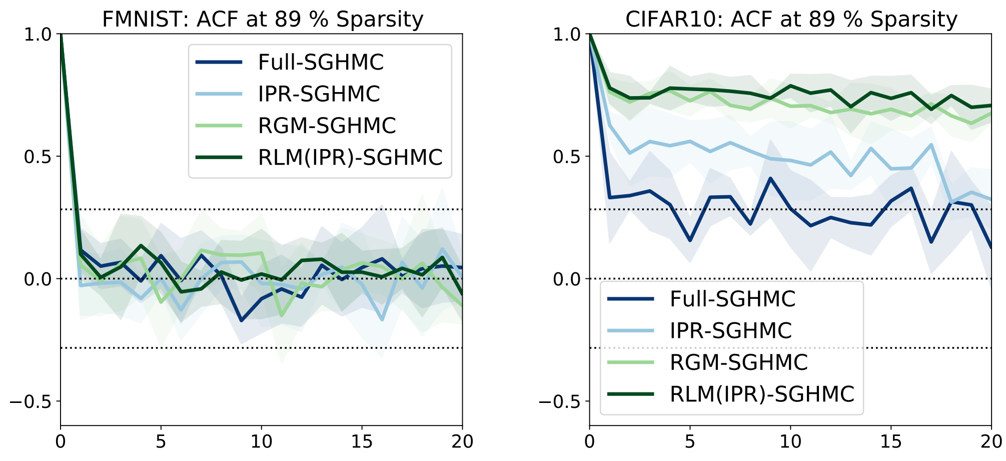


Figure 5.7. ACF plots for FMNIST and CIFAR10. The plots correspond to the case when sparsity level is set to 89%. We also include the full model as a baseline.

in the full models on all three data sets. Second, we note that SGHMC with the IPR-derived sparse structures also consistently out-performs the final optimized parameter values found for those same structures for all data sets and at all sparsity levels.

However, the relationship between the performance of the optimized models and SGHMC in sparse substructures is more complex. As we can see, at these sparsity levels, SGHMC under-performs Full-OPT on both FMNIST and CIFAR100. Note that the random layer-wise sparsity method shown uses 10 chains and, as observed previously, this leads to improvements in performance over the IPR-based SGHMC approach at lower sparsity levels. This approach does at least slightly out-perform optimization applied to the full model on all data sets. Finally, we observe that the IPR-based SGHMC method is able to out-perform the optimization-based full model on the CIFAR10 data set even at the highest level of sparsity. The key takeaway here is that we can obtain useful trade-offs by choosing parallel chain SGHMC on sparse substructures against SGHMC/SGD on full models, leading to train and test time savings as well as storage savings.

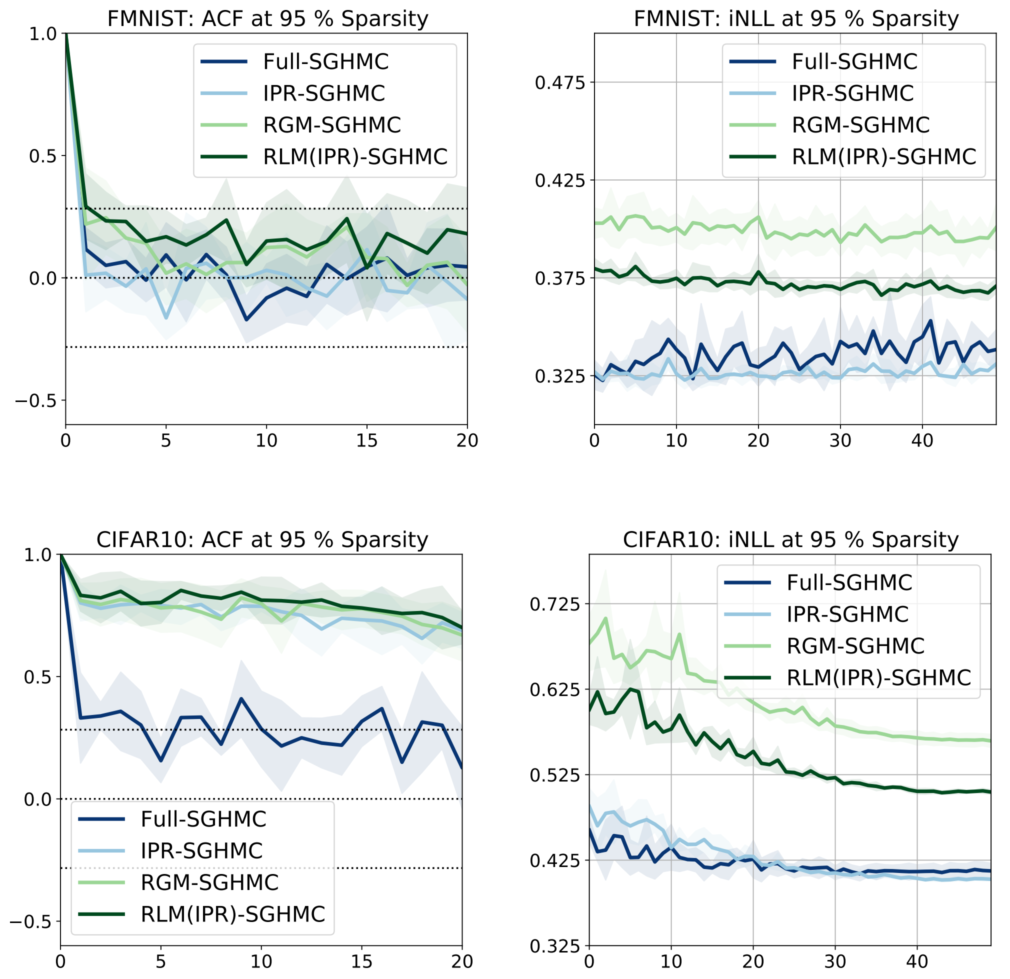


Figure 5.8. The top two figures show ACF and iNLL for FMNIST while the bottom two figures show ACF and iNLL for CIFAR10. The plots correspond to the case when sparsity level is set to 95%. We also include the full model as a baseline.

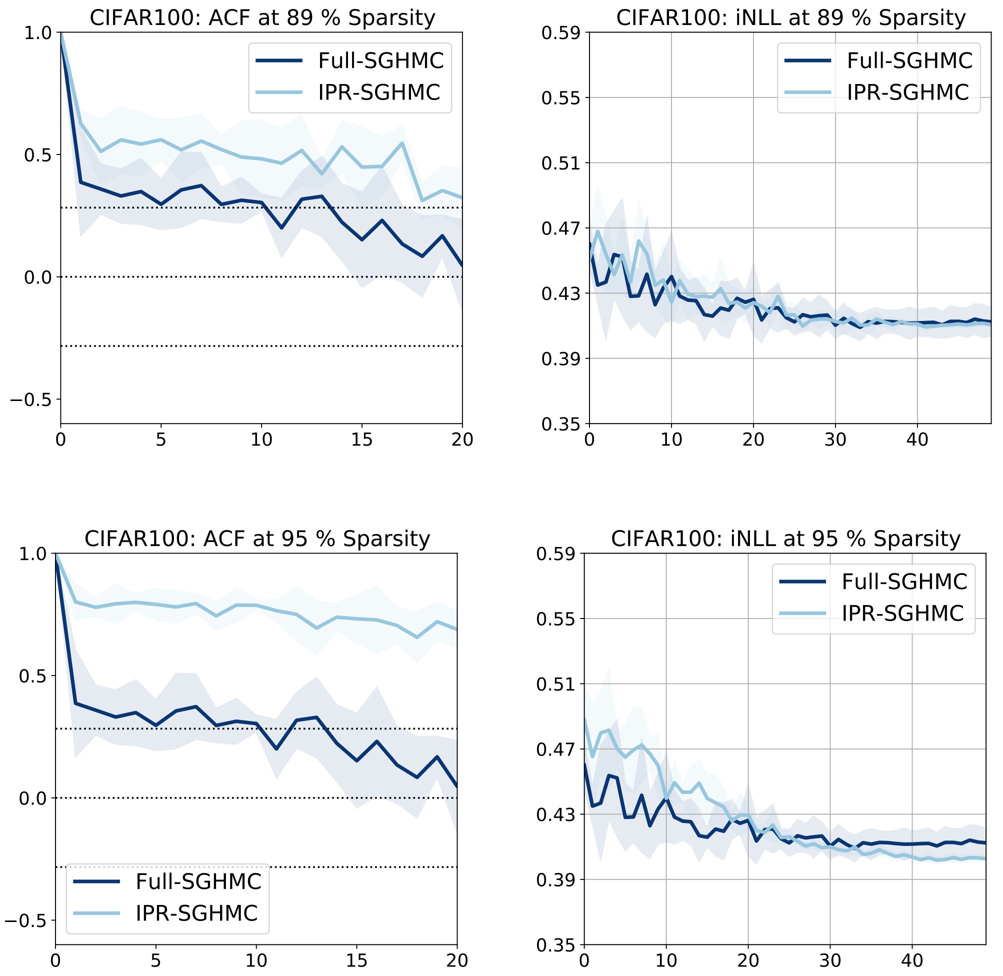


Figure 5.9. The top two figures show ACF and iNLL for CIFAR100 for a sparsity level of 89% while the bottom two figures show ACF and iNLL for CIFAR100 for a sparsity level of 95%. We also include a baseline with sparsity rate of 0%

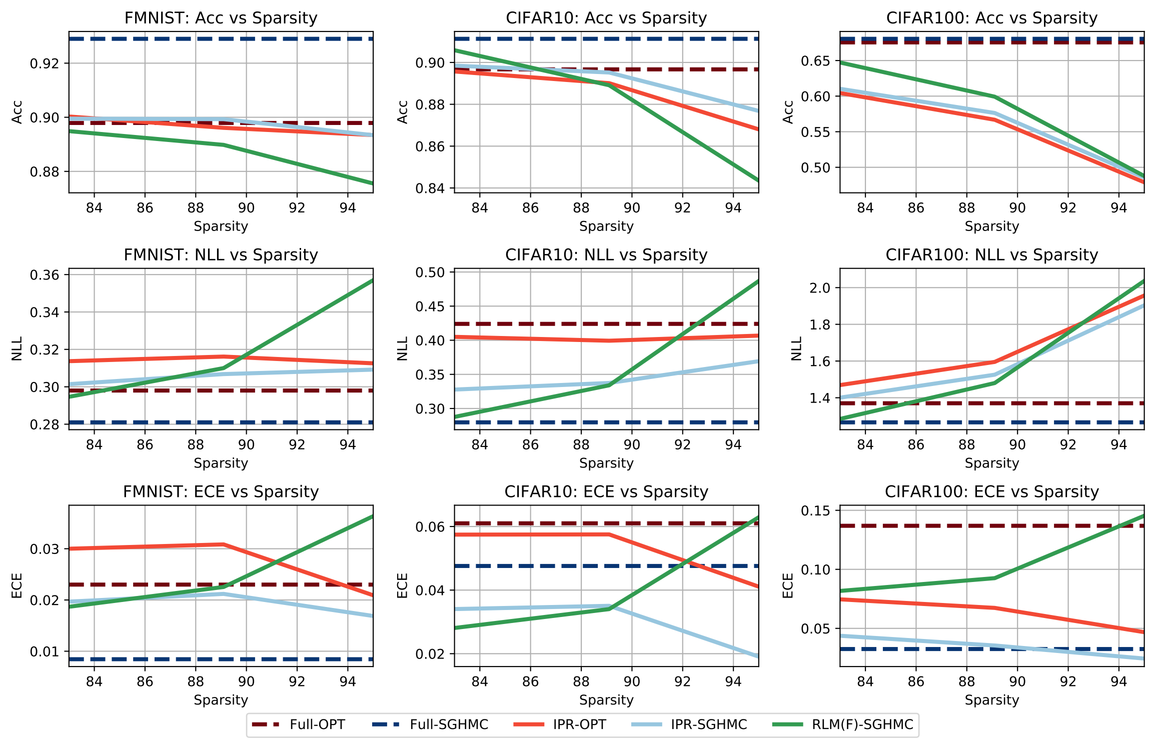


Figure 5.10. Performance of SGHMC and optimization-based learning on the FMNIST, CIFAR10 and CIFAR100 datasets.

5.3 Conclusions

In this chapter, we have explored the open question of how stochastic gradient MCMC methods perform when restricted to performing inference within different types of sparse sub-structures. We derive multiple conclusions from our experimental results. First, from the standpoint of classification performance, our results show that both single chains run within single substructures derived using iterative pruning methods and multiple chains run on multiple randomly selected sparse substructures can lead to effective trade-offs between posterior storage requirements, prediction time, and predictive performance compared to a full posterior ensemble for some model/data set combinations.

Perhaps the most surprising result is that at lower levels of sparsity, running multiple MCMC chains on randomly selected sub-structures performs very similarly to iterative pruning approaches that can not be parallelized, and thus require drastically higher training times. However, iterative pruning-based structures do appear to have an edge on randomly selected sparse structures in terms of predictive performance at higher sparsity rates. They also appear to exhibit better mean expected sample size and require less burn-in time. A key direction for future research is the development and evaluation of optimization methods for selecting sparse structures with more modest run times than iterative pruning strategies. Additional future directions include investigating the compression of sparse posterior ensembles using distillation methods and considering the composition of sparse substructures with subspace inference to further improve sampling efficiency.

CHAPTER 6

POST-HOC LOSS-CALIBRATION FOR BAYESIAN NEURAL NETWORKS

Bayesian decision theory provides an elegant framework for acting optimally under uncertainty when tractable posterior distributions are available. Modern Bayesian models, however, typically involve intractable posteriors that are approximated with surrogates. This difficulty has engendered loss-calibrated techniques that aim to learn posterior approximations that favor high-utility decisions. In this chapter, focusing on Bayesian neural networks, we develop methods for correcting approximate posterior predictive distributions, encouraging them to prefer high-utility decisions. In contrast to previous work, our approach is agnostic to the choice of the approximate inference algorithm, allows for efficient test time decision-making through amortization, and empirically produces higher quality decisions. We demonstrate the effectiveness of our approach through controlled experiments spanning a diversity of tasks and datasets.

The rest of the chapter is structured as follows. In Section 6.1 we introduce our post-hoc loss correction framework. Next, in Section 6.2 we present a comprehensive set of experiments and results demonstrating the utility of our framework and compare it against various baselines. Finally, we provide a discussion and a set of potential future directions in Section 6.3 to conclude this chapter.

Bibliographical note: This chapter is adapted from Vadera et al. [2021].

6.1 Framework

We begin with the assumption that we have access to a calibration dataset $\mathcal{D}' := \{\mathbf{x}_n\}_{n=1}^{N}$ and that we can evaluate the posterior predictive distribution, under some approximation to the posterior, at all $\mathbf{x}_n \in \mathcal{D}'$. The log conditional gain on $\mathcal{D}'$ is,

$$\begin{aligned} &\log \mathcal{G}(\mathbf{h} = \mathbf{c} \mid \mathcal{D}') = \\ &\quad \sum_{n=1}^{N} \log \int_{y} u(h = c_n, y_n = y) p(y_n = y \mid \mathbf{x}_n, \mathcal{D}, \theta^0) dy, \end{aligned} \tag{6.1}$$

where $\mathbf{c} = \{c_n\}_{n=1}^{N}$, and $c_n = \arg\max_{h \in \mathcal{A}} \mathcal{G}(h \mid \mathbf{x_n})$. Recall that

If we had access to the *true* posterior predictive distribution, guarantees from Bayesian decision theory ensure that the decisions c_n are optimal. However, for BNNs we only have access to potentially crude approximations to the posterior and c_n are no longer guaranteed to be optimal. To address this, we introduce an utility aware correction, $q(y_n \mid \mathbf{x}_n, \lambda)$ to the (approximate) posterior predictive distribution $p(y_n \mid \mathbf{x}_n, \mathcal{D}, \theta^0)$ evaluated at $\mathbf{x}_n \in \mathcal{D}'$. The correction is parameterized by a set of learnable parameters, λ. In our experiments, we use a neural network to parameterize q and λ corresponds to the weights of that network. We observe that the log conditional gain can be expressed as a function of λ,

$$\begin{aligned} &\log\ \mathcal{G}(\mathbf{h} = \mathbf{c} \mid \mathcal{D}'; \lambda) = \\ &\sum_{n=1}^{N} \log \mathbb{E}_{q(y_n = y | \mathbf{x}_n, \lambda)} \left[\frac{p(y_n = y | \mathbf{x_n}, \mathcal{D}, \theta^0) u(h = c_n, y_n = y)}{q(y_n = y | \mathbf{x}_n, \lambda)} \right], \end{aligned} \tag{6.2}$$

and is lower bounded by,

$$\begin{aligned} \mathcal{U}(\lambda, \mathbf{c}, \mathcal{D}') = &\sum_{n=1}^{N} \mathbb{E}_{q(y_n | \mathbf{x}_n, \lambda)} \left[\log u(c_n, y_n) \right] \\ &- \mathrm{KL}\big[q(y_n | \mathbf{x}_n, \lambda) || p(y_n | \mathbf{x}_n, \mathcal{D}, \theta^0) \big], \end{aligned} \tag{6.3}$$

where the bound $\log\, \mathcal{G}(\mathbf{h}=\mathbf{c} \mid \mathcal{D}'; \lambda) \geq \mathcal{U}(\lambda, \mathbf{c}; \mathcal{D}')$ follows from Jensen's inequality. In the following subsection, we provide a detailed derivation of our post-hoc correction objective.

6.1.1 Derivation of the post-hoc correction objective

We begin our derivation with the definition of log conditional gain as follows:

$$\begin{aligned}
&\log \mathcal{G}(\mathbf{h}=\mathbf{c} \mid \mathcal{D}'; \lambda) \\
&= \sum_{n=1}^{N} \log \int_y \Bigg(u(h=c_n, y_n=y) q(y_n=y|\mathbf{x_n}, \lambda) \\
&\qquad\qquad \times \frac{p(y_n=y|\mathbf{x_n}, \mathcal{D}, \theta^0)}{q(y_n=y|\mathbf{x_n}, \lambda)} dy \Bigg) \\
&= \sum_{n=1}^{N} \log \int_y \Bigg(q(y_n=y|\mathbf{x_n}, \lambda) \\
&\qquad\qquad \times \left(\frac{p(y_n=y|\mathbf{x_n}, \mathcal{D}, \theta^0) u(h=c_n, y_n=y)}{q(y_n=y|\mathbf{x_n}, \lambda)} \right) \Bigg) dy \\
&= \sum_{n=1}^{N} \log \mathbb{E}_{q(y_n=y|\mathbf{x_n},\lambda)} \left[\frac{p(y_n=y|\mathbf{x_n}, \mathcal{D}, \theta^0) u(h=c_n, y_n=y)}{q(y_n=y|\mathbf{x_n}, \lambda)} \right]
\end{aligned}$$

Now, using Jensen's inequality, we obtain,

$$\begin{aligned}
\log \mathcal{G}(\mathbf{h}=\mathbf{c}|\mathbf{X}) &\geq \sum_{n=1}^{N} \mathbb{E}_{q(y_n|\mathbf{x_n},\lambda)} \left[\log \left(\frac{p(y_n|\mathbf{x_n}, \mathcal{D}, \theta^0) u(c_n, y_n)}{q(y_n|\mathbf{x_n}, \lambda)} \right) \right] \\
&= \sum_{n=1}^{N} \mathbb{E}_{q(y_n|\mathbf{x_n},\lambda)} \left[\log u(c_n, y_n) \right] \\
&\quad - \mathrm{KL}(q(y_n|\mathbf{x_n}, \lambda) || p(y_n|\mathbf{x_n}, \mathcal{D}, \theta^0)) \\
&:= \mathcal{U}(\lambda, \mathbf{c}; \mathcal{D}') \qquad (6.4)
\end{aligned}$$

We learn $q(\cdot \mid \cdot, \lambda)$ by maximizing $\mathcal{U}(\lambda, \mathbf{c}; \mathcal{D}')$ with respect to λ and $\mathbf{c}$. Our algorithm proceeds in a coordinate ascent fashion by alternating between fixing $\mathbf{c}$ and taking a gradient step in the direction maximizing $\mathcal{U}(\lambda, \mathbf{c}; \mathcal{D}')$ with respect to λ and then fixing λ and maximizing $\mathbf{c}$. We limit our attention to finite discrete-valued decision problems prevalent in classification settings. For these problems, we are able to trivially maximize $\mathbf{c}$ given λ by enumerating the expected utility of all decisions and selecting the highest utility decision. In the next subsection, we derive the variational gap between the log conditional gain and the lower bound.

6.1.2 The variational gap

Consider a single data instance $\mathbf{x}_n \in \mathcal{D}'$. The gap between the log conditional gain and the lower bound then is,

$$\begin{aligned}
\log \mathcal{G}(h = c_n \mid \mathcal{D}'; \lambda) - \mathcal{U}(\lambda, c_n, \mathcal{D}') &= \log \int u(h = c_n, y_n = y) p(y_n | \mathbf{x_n}, \mathcal{D}, \theta^0) dy - \mathcal{U}(\lambda, c_n, \mathcal{D}') \\
&= \log \int u(h = c_n, y_n = y) p(y_n | \mathbf{x_n}, \mathcal{D}, \theta^0) dy - \int q(y_n = y | \mathbf{x}_n, \lambda) \log u(h = c_n, y_n = y) dy \\
&+ \int q(y_n = y | \mathbf{x}_n, \lambda) \log q(y_n = y | \mathbf{x}_n, \lambda) dy - \int q(y_n = y | \mathbf{x}_n, \lambda) \log p(y_n | \mathbf{x_n}, \mathcal{D}, \theta^0) \\
&= \log \int u(h = c_n, y_n = y) p(y_n | \mathbf{x_n}, \mathcal{D}, \theta^0) dy \\
&+ \int q(y_n = y | \mathbf{x}_n, \lambda) \log \frac{q(y_n = y | \mathbf{x}_n, \lambda)}{p(y_n | \mathbf{x_n}, \mathcal{D}, \theta^0) u(h = c_n, y_n = y)} dy \\
&= \log Z_n + \int q(y_n = y | \mathbf{x}_n, \lambda) \log \frac{q(y_n = y | \mathbf{x}_n, \lambda)}{p(y_n | \mathbf{x_n}, \mathcal{D}, \theta^0) u(h = c_n, y_n = y)} dy \\
&= \int q(y_n = y | \mathbf{x}_n, \lambda) \log Z_n dy + \int q(y_n = y | \mathbf{x}_n, \lambda) \log \frac{q(y_n = y | \mathbf{x}_n, \lambda)}{p(y_n | \mathbf{x_n}, \mathcal{D}, \theta^0) u(h = c_n, y_n = y)} dy \\
&= \int q(y_n = y | \mathbf{x}_n, \lambda) \log \frac{q(y_n = y | \mathbf{x}_n, \lambda) Z_n}{p(y_n | \mathbf{x_n}, \mathcal{D}, \theta^0) u(h = c_n, y_n = y)} dy \\
&= \mathrm{KL}\left[q(y_n | \mathbf{x}_n, \lambda) || \frac{p(y_n | \mathbf{x}_n, \mathcal{D}, \theta^0) u(h = c_n, y_n)}{Z_n} \right].
\end{aligned} \tag{6.5}$$

Thus, the variational gap between the log conditional gain and the lower bound over the entire dataset can be computed by summing over all datapoints, and is given by

$$\begin{aligned}&\log\mathcal{G}(\mathbf{h} = \mathbf{c} \mid \mathcal{D}'; \lambda) - \mathcal{U}(\lambda, \mathbf{c}, \mathcal{D}') \\ &\quad = \sum_{\mathbf{x}_n \in \mathcal{D}'} \mathrm{KL}\left[q(y_n|\mathbf{x}_n, \lambda) || \frac{p(y_n|\mathbf{x}_n, \mathcal{D}, \theta^0) u(c_n, y_n)}{Z_n} \right],\end{aligned} \tag{6.6}$$

where $Z_n = \mathbb{E}_{p(y_n|\mathbf{x}_n, \mathcal{D}, \theta^0)}[u(c_n, y_n)]$. Importantly, this lends us further insights into the optimization problem. For a fixed $\mathbf{c}$, maximizing Eq. (6.3) is equivalent to minimizing the KL divergence between q and the original posterior predictive distribution scaled by the utility function, pointwise over the calibration dataset. This further highlights a key aspect of the proposed approach, it corrects the (*typically*) low-dimensional posterior predictive distribution rather than the unwieldy, high-dimensional BNN posterior. The lower bound Eq. (6.3) also lends itself to an intuitive interpretation. The first term guides $q(\cdot \mid \cdot, \lambda)$ to higher utility decisions, while the second Kullback-Leibler divergence term encourages $q(\cdot \mid \cdot, \lambda)$ to be close to the approximate posterior predictive distribution in the KL sense.

Although nearly operational, two key challenges remain in applying the developed framework. The first stems from computational considerations necessary when working with large Bayesian models like BNNs. Posterior predictive distributions for BNNs need to be approximated via Monte Carlo simulations. Computation and storage cost of Monte Carlo approximations grow linearly with the number of samples and can be prohibitive for large networks. The other challenge stems from user preferences typically being expressed as cost functions [Berger, 1985, Kuśmierczyk et al., 2019] rather than utility functions, and yet our development thus far has dealt exclusively with utility functions. We next describe strategies effective at alleviating both these concerns.

6.1.3 Practical considerations

6.1.3.1 Amortized posterior predictive distribution

We tackle the computational concerns associated with Monte Carlo approximations to the posterior predictive distribution by learning an amortized approximation [Balan et al., 2015, Vadera et al., 2020b]. We use the online distillation algorithm proposed by Balan et al. [2015], a special case of the general framework of Vadera et al. [2020b], and distill the posterior predictive distribution into a single "student" neural network model. This algorithm aims to minimize the Kullback-Leibler (KL) divergence between $p(y_n \mid \mathbf{x}_n, \mathcal{D}, \theta^0)$ and a student network $S(y_n \mid \mathbf{x}_n, \omega)$, parameterized by ω for $\mathbf{x}_n \in \mathcal{D}'$. The online nature of this algorithm allows us to amortize the computation of posterior predictive distribution, without having to instantiate numerous posterior samples. Once we have trained the student model, we can use it as a drop-in replacement for the posterior predictive distribution in Eq. (6.3),

$$\begin{aligned}\mathcal{U}^s(\lambda, \mathbf{c}, \mathcal{D}') = &\sum_{n=1}^{N} \mathbb{E}_{q(y_n|\mathbf{x_n},\lambda)} \left[\log u(c_n, y_n)\right] \\ &- \mathrm{KL}(q(y_n \mid \mathbf{x_n}, \lambda)||S(y_n \mid \mathbf{x_n}, \omega)).\end{aligned} \tag{6.7}$$

6.1.3.2 Decision cost v/s utilities

In practical applications it is common to have user preferences encoded as decision costs rather than utilities. We follow Kuśmierczyk et al. [2019], to translate between costs and utilities. Let us denote the decision cost function as $\ell(h, y)$, where h again denotes the decision and y denotes the predicted class. W can re-write the utility function as $u(h, y) = M - \ell(h, y)$, where $M \geq \sup_{h,y} \ell(h, y)$. By substituting this in Eq. (6.3) we obtain,

$$\begin{aligned}\mathcal{L}(\lambda, \mathbf{c}; \mathcal{D}') = &\sum_{n=1}^{N} \mathbb{E}_{q(y_n|\mathbf{x_n},\lambda)} \Big[\log \big(M - \ell(c_n, y)\big)\Big] \\ &- \mathrm{KL}(q(y_n|\mathbf{x_n}, \lambda)||p(y_n|\mathbf{x_n}, \mathcal{D}, \theta^0)),\end{aligned} \tag{6.8}$$

and the analogous amortized variant is given by,

$$\begin{aligned}\mathcal{L}^s(\lambda, \mathbf{c}; \mathcal{D}') = &\sum_{n=1}^{N} \mathbb{E}_{q(y_n|\mathbf{x_n},\lambda)} \Big[\log\big(M - \ell(c_n, y)\big)\Big] \\ &- \mathrm{KL}(q(y_n|\mathbf{x_n}, \lambda)||S(y_n|\mathbf{x_n}, \omega)).\end{aligned} \tag{6.9}$$

Further performing a first order Taylor series expansion about M [Kuśmierczyk et al., 2019, Lacoste-Julien et al., 2011] we obtain,

$$\begin{aligned}\mathcal{L}^s(\lambda, \mathbf{c}; \mathcal{D}') \approx &\sum_{n=1}^{N} \mathbb{E}_{q(y_n|\mathbf{x_n},\lambda)} \left[\log M - \frac{\ell(c_n, y)}{M}\right] \\ &- \mathrm{KL}(q(y_n|\mathbf{x_n}, \lambda)||S(y_n|\mathbf{x_n}, \omega)),\end{aligned} \tag{6.10}$$

Noting that $\mathbb{E}_{q(y_n|\mathbf{x_n},\lambda)}[\log M]$ is constant with respect to λ and $\mathbf{c}$, we arrive at,

$$\begin{aligned}\tilde{\mathcal{L}}^s(\lambda, \mathbf{c}; \mathcal{D}') = - &\sum_{n=1}^{N} \mathbb{E}_{q(y_n|\mathbf{x_n},\lambda)} \left[\frac{\ell(c_n, y)}{M}\right] \\ &- \mathrm{KL}(q(y_n|\mathbf{x_n}, \lambda)||S(y_n|\mathbf{x_n}, \omega)).\end{aligned} \tag{6.11}$$

If we have access to the original posterior predictive distribution $p(y_n|\mathbf{x_n}, \mathcal{D}, \theta^0))$, the analogous objective is,

$$\begin{aligned}\tilde{\mathcal{L}}(\lambda, \mathbf{c}; \mathcal{D}') = - &\sum_{n=1}^{N} \mathbb{E}_{q(y_n|\mathbf{x_n},\lambda)} \left[\frac{\ell(c_n, y)}{M}\right] \\ &- \mathrm{KL}(q(y_n|\mathbf{x_n}, \lambda)||p(y_n|\mathbf{x_n}, \mathcal{D}, \theta^0))).\end{aligned} \tag{6.12}$$

Our experiments maximize either $\tilde{\mathcal{L}}^s(\lambda, \mathbf{c}; \mathcal{D}')$ or $\tilde{\mathcal{L}}(\lambda, \mathbf{c}; \mathcal{D}')$ depending on the experimental setup. An algorithm giving an overview of our approach is presented in Algorithm 3.

With the description of our method complete, we reemphasize the distinct advantages provided by it. Observe that we only require that we are either able to evaluate the posterior predictive distribution or an amortized approximation to it on $\mathcal{D}'$. We

Algorithm 3 Post-hoc loss correction algorithm

```
Require: Amortized posterior approximation $S(.|., \omega)$, Calibration dataset $\mathcal{D}'$, mini-batch size $B$, loss-calibrated model $q(.|., \lambda)$, number of training iterations $T$, initialization $\lambda_0$, supremum of the loss function $\ell$, $M$.
 1: Initialize the loss calibrated model by setting its parameters to $\lambda_0$.
 2: for t $\in [1, \ldots, T]$ do
 3:     Draw a mini-batch $\mathcal{B} \subset \mathcal{D}'$ of size $B$.
 4:     for each data point $\mathbf{x}_b \in \mathcal{B}$ do
 5:         Compute $c_b = \arg\min_c \int_{y_b} \ell(c, y_b) q(y_b \mid \mathbf{x}_b, \lambda_t) dy_b$
 6:     end for
 7:     Define $\tilde{\mathcal{L}}(\lambda, \mathbf{c}; \mathcal{B}) = -\sum_{b=1}^{B} \mathbb{E}_{q(y_b|\mathbf{x}_b,\lambda)} \left[ \frac{\ell(c_b, y_b)}{M} \right] - \mathrm{KL}(q(y_b|\mathbf{x}_b, \lambda) || S(y_b|\mathbf{x}_b, \omega)))$
 8:     Update $\lambda_{t+1} \leftarrow \mathrm{SGDUpdate}(\lambda_t, \nabla_\lambda \tilde{\mathcal{L}}(\lambda, \mathbf{c}, \mathcal{B}))$
 9: end for
10: return Optimized parameters of the loss calibrated model, $\lambda_T$.
```

remain agnostic and make no assumptions about how the posterior or the posterior predictive distributions were computed. Moreover, learning the corrections, $q(\cdot \mid \cdot, \lambda)$, involves optimizing Eq. (6.11) or Eq. (6.12) and is no more expensive than training standard deep neural networks. Finally, at a test point, $\mathbf{x}_*$ the expected cost associated with a decision h is $\sum_{k=1}^{C} \ell(h, y = k) q(y = k \mid x_*, \lambda)$. Computing this expected cost involves a *single* forward pass through q. Our framework, thus, amortizes test time decision-making. This leads to significant speed-ups over existing loss-calibrated inference approaches, which must first compute the posterior predictive distribution by performing an expensive Monte Carlo integration over the corrected posterior before making decisions. Test time amortization allows our method to be used in applications that demand real-time decision-making. We summarize our approach's similarities and differences to relevant work in Table 6.1.

6.2 Experiments

In this section, we compare our proposed method against relevant baselines across a diverse range of applications. Broadly, we divide our experiments into three major categories to target three practical scenarios — 1) Decision-making in poor data

Table 6.1. Overview of related loss-calibrated methods. *Inference Agnostic*: Method does not modify or make assumptions about the posterior inference algorithm. *Scalable*: Method scales to modern Bayesian neural networks learned from large data. *Amortized Decisions*: Method does not require multiple forward passes for test time decision-making.

	Inference Agnostic	Scalable	Amortized Decisions
Lacoste-Julien et al. [2011]	✗	✗	✗
Cobb et al. [2018a]	✗	✓	✗
Kuśmierczyk et al. [2019]	✗	✓	✗
Kuśmierczyk et al. [2020]	✓	✗	✓
Ours	✓	✓	✓

quality regimes, 2) Decision-making with a reject option (also known as *selective classification*), 3) Decision-making under real-time constraints.

Throughout this section, we experiment with fully-connected and ResNet18 [Krizhevsky et al., 2012] architectures. We test our methods on data from MNIST [LeCun, 1998], CIFAR10 [Krizhevsky et al., 2009], and the challenging CamVid [Brostow et al., 2008b] dataset. We demonstrate that our posterior correction consistently improves the quality of decisions when used in conjunction with popular BNN inference algorithms — black-box variational inference (BBVI) [Blundell et al., 2015], stochastic gradient Hamiltonian Monte-Carlo (SGHMC) [Chen et al., 2014a], and Kronecker-factored Laplace approximation (KFAC-Laplace) [Ritter et al., 2018]. Noting that SGHMC [Yao et al., 2019] typically provides a more faithful approximation to the BNN posterior, we restrict ourselves to SGHMC for real data experiments. To compare against a loss-calibrated inference procedure, we develop a loss-calibrated variant of SGHMC. Following Lacoste-Julien et al. [2011] we define the following utility scaled posterior,

$$\tilde{p}(\theta|\mathcal{D}, u) \propto p(\theta|\mathcal{D})\mathcal{G}(\mathbf{h}^*|\mathcal{D}). \tag{6.13}$$

The loss-calibrated stochastic gradient HMC (LC-SGHMC) algorithm then proceeds by sampling from this scaled posterior using SGHMC. Given that SGHMC is typically more accurate than competing variational methods, we view LC-SGHMC as a strong loss-calibrated inference baseline.

6.2.1 Synthetic Data Experiments

We begin with experiments on synthetic data, employing fully connected architectures and three popular inference techniques, BBVI with local reparameterizations [Kingma et al., 2015], SGHMC, and KFAC-Laplace.

6.2.1.1 Experimental setup

We construct a two-dimensional, two class data set with class imbalance. We generate data from the two classes by sampling isotropic Gaussian distributions with means $[-1,-1]$ and $[+1,+1]$. For training, we use 90 data instances from the negative class and 10 data instances from the positive class. We resample the Gaussian distributions to create a test set, which again contains 90 negative examples and 10 positive ones. For calibration data, we uniformly sample the two dimensional space to generate 500 unlabeled data instances. We repeat this procedure ten times and generate ten training and calibration datasets. Fig. 6.1 visualizes one of these ten datasets. For each dataset, we learn a 50 unit, single hidden layer, multi-layer perceptron with ReLU activations using BBVI, SGHMC, and KFAC-Laplace and use a 100 sample Monte Carlo approximation to compute the corresponding posterior predictive distributions. We employ the following decision-cost function,

$$\ell(c, y) = \begin{cases} 0, & \text{for } y = c \\ 1, & \text{for } y \neq c, y = \text{positive} \\ 0.1, & \text{for } y \neq c, y = \text{negative} \end{cases},$$

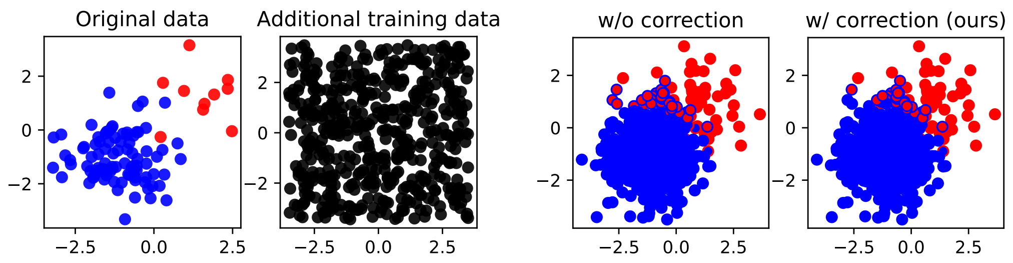


Figure 6.1. Synthetic Data (Top). Labeled training data ($\mathcal{D}$) is shown in the left plot and unlabeled calibration data ($\mathcal{D}'$) is shown in the right plot. Blue markers represent the negative class and red markers represent the positive class. **Decision Visualization (Bottom)**. A visual comparison of decision-making using BBVI without correction, and with our correctional approach for a single trial, on test data drawn from the same distribution as the training data $\mathcal{D}$. The edge colors indicate ground truth classes, while the face colors indicate the predicted classes. Consistency between edge and face colors indicate correct predictions.

which encourages decisions that minimize false negative errors for the minority class — often a desirable property in practice. In this experiment, we learn the corrections by maximizing Eq. (6.12).

For the SGHMC implementation in this experiment, we use a fixed learning rate of 0.1, a momentum of 0.5 and prior precision of 1.0. We run a burn-in phase of 300 iterations, and collect total 100 parameter samples with a thinning interval of 50 iterations. After this, we obtain the monte carlo approximation to the posterior predictive distribution $p(y|\mathbf{x}, \mathcal{D})$ on the additional training data points ($\mathcal{D}'$). For doing our post hoc correction, we use an MLP with 50 hidden units (as with the original model) and optimize the objective shown in Eq. (6.11) using Adam optimizer with a learning rate of 0.1 for 500 training iterations.

For BBVI implementation in this experiment, we set an identity gaussian distribution as our prior and maximize the evidence lower bound (ELBO) of our MLP-BNN using Adam optimizer with a learning rate of 0.01 over 5000 iterations. We use the local-reparameterization trick Kingma et al. [2015] to produce low variance stochastic gradients. Next, we again collect a total of 100 parameter samples and compute the Monte Carlo approximation of the posterior predictive distribution $p(y|\mathbf{x}, \mathcal{D})$ on the additional training data points ($\mathcal{D}'$). For doing our post hoc correction, we use an MLP with 50 hidden units (as with the original model) as our $q(.)$ model, and optimize the objective shown in Eq. (6.11) using Adam optimizer with a learning rate of 0.1 for 500 training iterations.

For KFAC-Laplace we perform a Laplace approximation about the maximum-a-posteriori (MAP) solution. As in standard Laplace approximation, the approximate posterior is represented by a Gaussian centered at the MAP solution with its covariance set to the inverse of the Hessian. We find the MAP solution by using Adam with a learning rate of 0.01 to maximize the negative log posterior. Following Ritter et al. [2018] we use a block-diagonal, Kronecker-factored Hessian.

Finally, note that there are 100 training points in the original training set ($\mathcal{D}$), 500 additional unlabeled data points for our posterior correction ($\mathcal{D}'$), and 100 held out data points for testing from the same data distribution as $\mathcal{D}$.

6.2.1.2 Results

We compute the average decision cost under our cost-function for each of the three inference algorithms with and without our post-hoc correction on the test set. Table 6.2 summarizes our results, where the error bars stem from having repeated the experiment on the ten randomly generated training and calibration datasets. Our post-hoc correction results in test decisions with lower decision costs compared to the decisions produced by the uncorrected variants. The costs are only marginally lower in this synthetic example, where the approximations to the posterior are likely already good. In the following, we will see that the decision costs can be substantially lower in more challenging scenarios. In subsequent experiments, we solely rely on stochastic gradient HMC algorithms for approximating the Bayesian neural network posterior.

Table 6.2. Results on synthetic data. Test decision costs with and without post-hoc correction over 10 replicates. Post-hoc correction consistently provides lower cost decisions. Results presented as mean ± std. dev.

	W/O post-hoc correction	W/ post-hoc correction (ours)
VI	0.019 ± 0.011	0.016 ± 0.010
SGHMC	0.018 ± 0.008	0.017 ± 0.009
KFAC-Laplace	0.021 ± 0.007	0.018 ± 0.008

6.2.2 Selective Classification

Next, we consider the problem of selective classification, wherein the goal is to classify a data instance into one of C classes or choose not to classify and instead refer it to an oracle. The corresponding decision problem thus involves selecting one of $C + 1$ decisions for each data instance. By adjusting the cost of a referral, the decision-making system can trade erroneous decisions for potentially expensive oracle feedback. Different users of such a system will likely prefer different trade-offs and as a result choose different referral costs. We however have no reason to believe that the different users would have different posterior beliefs. Since our method does not involve relearning the posterior beliefs when faced with changing cost functions, it is well suited for such selective classification problems.

6.2.2.1 Experimental setup

We use the CIFAR10 [Krizhevsky et al., 2009] dataset along with SGHMC trained Bayesian ResNet18 [He et al., 2016a] networks. To make the problem more challenging and encourage referrals, we contaminate the data via an additional data transformation. Under this contamination, we rotate each image in the dataset by an angle sampled uniformly at random from $[-30°, 30°]$. Next, following Murphy [2012] (section 5.7.1.2), we define our selective classification decision cost function as,

$$\ell(c, y) = \begin{cases} 0, & \text{for } y = c \\ 1, & \text{for } y \neq c \\ r, & \text{for } c = \text{referral.} \end{cases}$$

Here, r denotes the cost of a referral. With this setup we examine a) the effectiveness of our method as a function of r, b) whether using the amortized posterior predictive distribution S (maximizing Eq. (6.11)) adversely affects performance when compared to the non-amortized version (maximizing Eq. (6.12)), and c) the computational cost of learning post-hoc corrections under multiple decision-cost functions.

In this experiment, we employ two methods for approximating the posterior predictive distribution $p(y|\mathbf{x}, \mathcal{D}, \theta^0)$: using a student model and pre-computing the posterior predictive distribution using the samples from SGHMC. For both cases, we generate the additional unlabeled training data $\mathcal{D}'$ by applying the same random transformation on the original CIFAR10 data set, and generate 10 copies for every example by randomly rotating the image as described earlier. In the case where we use the student model, we distill the posterior predictive distribution using the approach of Balan et al. [2015] and use the same $\mathcal{D}'$ for the distillation. Note that the approach by Balan et al. [2015] allows us to interleave the sampling from $p(\theta|\mathcal{D}, \theta')$ and distilling to a student model using an online approach.

Finally, once we obtain some form of approximation for the posterior predictive distribution, we optimize either Eq. (6.11) or, Eq. (6.12) depending on whether we have used the student or not. We use the same ResNet18 architecture for the $q(.)$ model. For the SGHMC chains, we use a momentum of 0.7, a fixed learning rate of 10^{-3}, and a prior precision of 5, and 1000 burn-in iterations (each iteration is a gradient step after a mini-batch). We run the SGMCMC sampling-distillation algorithm from Balan et al. [2015] for a total of 100 epochs, and collect a total of 30 samples at the end of each of the last 30 epochs. The student model has the same ResNet18 architecture, and is trained using SGD with a one-cycle learning rate schedule [Smith, 2017]. The peak learning rate is 0.1 and the weight decay factor is 5×10^{-4}. We warm start our $q(.)$ model with the student and train it for 100 epochs using an SGD optimizer with a learning rate of 10^{-3}.

For the case where we don't use a student model, we train our $q(.)$ model using SGD with a cyclic learning rate schedule, as it helps accelerate the training. The peak learning rate is 0.1. We use the equivalent number of training iterations as 100 epochs on the original data set $\mathcal{D}$. For sampling a mini-batch from $\mathcal{D}'$, we use sampling without replacement at every iteration. The mini-batch size for the entire experiment is set to 64. Also note that the SGHMC chain begins with a pretrained maximum-a-posteriori (MAP) solution. This pre-trained solution is obtained using an SGD optimizer with a one-cycle learning rate scheduler, with a peak learning rate of 0.05, with the same prior precision, and is trained for 100 epochs using $\mathcal{D}$.

6.2.2.2 Results

The results for the experiment are presented in Fig. 6.2. As we would expect, lower values of the referral cost r lead to more referrals and as a result, the models tend to make a decision only when they are very confident, leading to higher values of accuracy. Lower values of r also result in lower average decision cost, as the models tend to refer to the oracle more, and due to the lower value of referral cost, the average decision cost reduces. As we can see from the comparison, the decision cost as well as accuracy of our post-hoc corrections outperforms those obtained using SGHMC and LC-SGHMC across all values of referral cost r. Furthermore, we observe that the accuracy and decision cost values are not adversely affected by using S in place of the non-amortized posterior predictive distribution.

We also compare the training time of LC-SGHMC and our post-hoc corrections as a function of the number of decision-cost functions. The wall-clock times required by the

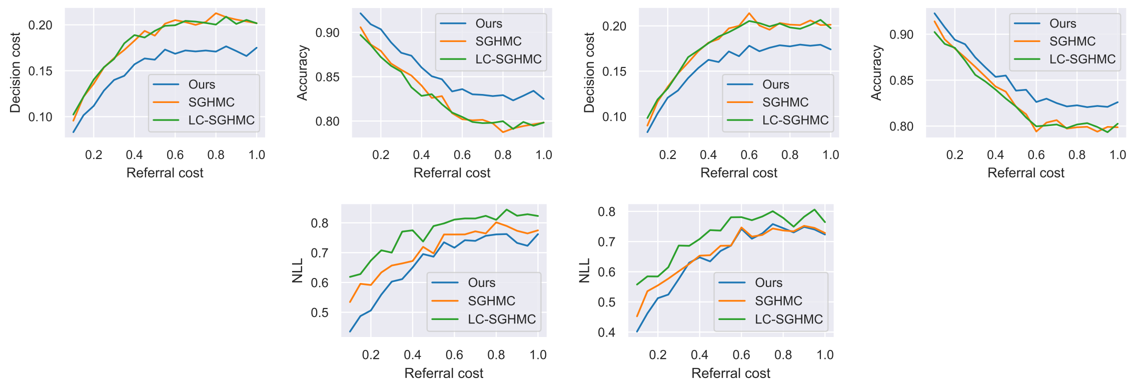


Figure 6.2. Selective classification. (Top Row) Average test decision costs and test accuracy as a function of referral cost on CIFAR10 using a Bayesian ResNet18 model. The two left plots show results without using an amortized posterior predictive distribution, S, and the two plots on the right display results when using S. Accuracy is determined on those test data points that are not referred to the oracle. We note that introducing the amortized approximation S does not adversely affect performance. (Bottom Row) NLL comparison of unreferred data points for different levels of referral cost for CIFAR10 data set using the same Bayesian ResNet18 model. The left plot represent results from the experiment not using the student S to approximate the posterior predictive distribution, while the right plot represents results from the experiment using the student S to approximate the posterior predictive distribution.

two approaches are shown in Fig. 6.3. All the experiments were run using the same GPU hardware (Nvidia Tesla V100) and under identical conditions for consistency in wall-clock time comparisons. We observe that our approach is significantly faster to train, and its computational cost grows at a slower rate with increasing number of cost functions.

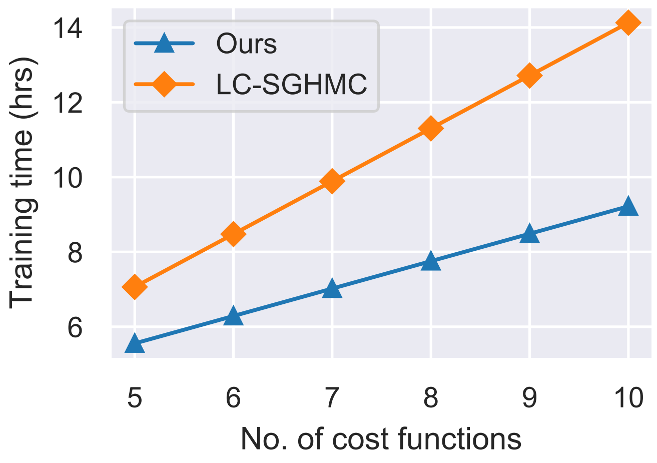


Figure 6.3. Training cost comparison. Wall-clock training time of our method and LC-SGHMC as a function of the number of decision cost functions.

Finally, the fact that the amortized posterior predictive distribution S does not adversely affect performance, is significant, in that it suggests that our approach would likely continue to scale with increasing network size, when storing and averaging over multiple Monte Carlo samples to compute the posterior predictive distribution is the bottleneck. Moving forward, we only consider experiments with the amortized posterior predictive S.

6.2.3 Decision-making under poor data quality

In the current era of big-data, it is not uncommon to have data sets of poor quality. In many settings, the data sets are labeled by crowdsourcing, as well as other automated techniques. These labeling techniques can often lead to noisy labels, and affecting downstream performance. Thus, it is important to understand how our method performs in this practical scenario. In practical scenarios, it is common to have asymmetric decision cost functions. This means that for making certain incorrect decisions, the cost can be higher or lower than the rest to encourage or discourage making those decisions. In this experiment, we also incorporate an asymmetric cost function similar to the one introduced earlier in the synthetic data experiments.

6.2.3.1 Experimental setup

We simulate label corruption on MNIST [LeCun, 1998] and CIFAR10 [Krizhevsky et al., 2009]. For each dataset, we switch the true labels of a proportion of the training set to labels sampled uniformly at random. For MNIST, we use a simple multi-layer perceptron architecture with one hidden layer of 200 units, while for CIFAR10, we use the ResNet18 architecture [He et al., 2016a]. Recall that SGLD can be derived from SGHMC as shown in Chapter 2, Section 2.1. For each data set, we pick two classes to which we assign higher importance, and thus assigning a lower cost to the mistakes which involve choosing these classes as decisions. In MNIST, we assign a higher importance to classes 3 and 8, while in CIFAR10, we assign a higher importance to classes automobile and trucks. We use SGLD [Welling and Teh, 2011] for sampling from the posterior as well as the utility scaled posterior distribution. For a point-estimated model baseline, we introduce the class-weighted SGD (CW-SGD) baseline. In CW-SGD, we use our standard log loss on the neural network model, but assign a higher weight to the classes of interest. This encourages the model to make fewer mistakes on classes of higher importance.

In this experiment, we use a ResNet18 model with the CIFAR10 data set and an MLP with a single hidden layer of 200 units (MLP200) for MNIST. We generate $\mathcal{D}'$ online, by adding random pixel noise from a $\mathcal{N}(0, 0.05)$ distribution. Since we use student model, we follow the same algorithm and subsequent post-hoc correction as described earlier in. Student model S and our $q(.)$ model follow the same architecture as of the original model.

For the ResNet18-CIFAR10 combination, we use SGLD with a fixed learning rate of 2×10^{-5} and a prior precision of 50. The burn-in iterations is set to 10,000. We run the distillation algorithm to train the student model for 50 epochs, and collect 30 samples from the original model (also the teacher model here) from the end of the last 30 epochs. To train the student model, we use SGD with a cyclic learning rate schedule. The maximum learning rate in the schedule is 0.05, with a momentum of 0.9 and a weight decay factor of 10^{-4}. Once we obtain the student, we train our $q(.)$ model using the objective in Eq. (6.11). For training the $q(.)$ model, we use SGD with a cyclic learning rate schedule. The maximum learning rate in the schedule is 0.05, with a momentum of 0.9. Note that for running SGHMC on ResNet18-CIFAR10 combination, we start with a pre-trained solution. This pre-trained MAP solution is obtained by running our original teacher model using the same data set $\mathcal{D}$. For the pre-training step, we use SGD with a cosine annealing learning rate schedule with warm restarts. The initial learning rate in the schedule is 0.05, with a momentum of 0.9 and a weight decay factor of 5×10^{-4}. Each annealing phase is of 20 epochs, and we perform a total of 5 such phases.

For the MNIST-MLP200 combination, we use SGLD (obtained by setting $\alpha = 1$ in SGHMC, see Chapter 2, Section 2.1) with a fixed learning rate of 10^{-4} and a prior precision of 6. The burn-in iterations is set to 10,000. We run the distillation algorithm to train the student model for 100 epochs, and collect 30 samples from the original model (also the teacher model here) from the end of the last 30 epochs. To train the student model, we use SGD with a fixed learning rate of 10^{-3}, with a momentum of 0.9 and a weight decay factor of 10^{-4}. Once we obtain the student, we train our $q(.)$ model using the objective in Eq. (6.11). For training the $q(.)$ model, we use SGD with a fixed learning rate of 10^{-3}, with a momentum of 0.9. Note that for running SGLD on MNIST-MLP200 combination, we start with a pre-trained MAP solution as well. For the pre-training step, we use SGD with a cosine annealing learning with warm restarts. The initial learning rate in the schedule is 0.01, with a momentum of 0.9 and a weight decay factor of 1×10^{-4}. Each annealing phase is of 20 epochs, and we perform total 5 such phases.

The decision cost function for CIFAR10 is defined as shown below:

$$\ell(c, y) = \begin{cases} 0, & \text{for } y = c, \\ 0.7, & \text{for } y \neq c, \text{ and c} \in \{\text{automobile, truck}\} \\ 1.0, & \text{otherwise} \end{cases} \tag{6.14}$$

The decision cost function for MNIST is defined as shown below:

$$\ell(c, y) = \begin{cases} 0, & \text{for } y = c, \\ 0.7, & \text{for } y \neq c, \text{ and c} \in \{3,\ 8\} \\ 1.0, & \text{otherwise} \end{cases} \tag{6.15}$$

For the CW-SGD baseline, we assign a loss weight of 1.4 to the instance if the ground truth is either automobile or truck for CIFAR10 and if the ground truth either 3 or 8 for MNIST. The reason for this is that we want to our system to assign higher importance to these classes, and thus a mistake of making an incorrect decision with these ground truth classes should incur higher loss. Similarly, the decision cost function highlights the same fact, as it places a lesser penalty if our decision system predicts these important classes.

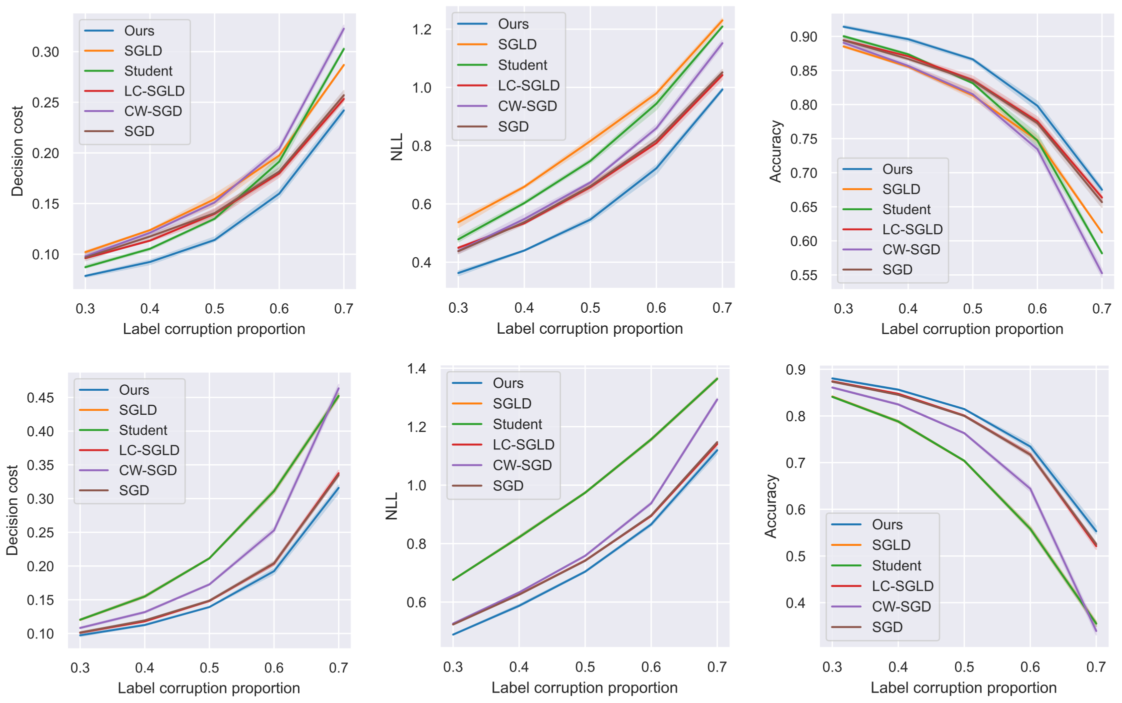


Figure 6.4. Decision-making under label corruption. The top row illustrates performance comparison for different levels of label corruption on CIFAR10 using a Bayesian ResNet18 model. The bottom row illustrates performance comparison for different levels of label corruption on MNIST using an MLP with a single hidden layer of 200 hidden units. The results are shown as mean $\pm$ std. dev. over 5 trials.

6.2.3.2 Results

We present the results of this experiment in Figure 6.4. For a comprehensive evaluation of performance, we vary the label corruption proportion between 0.3 and 0.7. For performance assessment, we look at the decision cost (↓), and accuracy (↑) on the standard test sets for each data set. While looking across the set of performance metrics, similar trends emerge. For lower levels of corruption, we notice that our post-hoc correction method performs similarly to LC-SGHMC and LC-SGLD, and marginally better than the uncorrected posterior predictive distribution and CW-SGD. However, as we increase the label corruption proportion to moderate levels, we observe that our method outperforms the baselines. Finally, with increasing corruption proportion, all methods are overwhelmed by the label noise, and we see a sharp dip in performance across all methods. We note that beyond achieving similar or lower decision costs, our approach also achieves higher accuracy and negative log-likelihoods than the competing methods. With these encouraging results in mind, we move towards our final experiment, which looks at a real-world data set and demonstrates the benefits of amortized decision-making.

6.2.4 Semantic scene segmentation

In this experiment, we consider the problem of semantic scene segmentation. Here, our goal is to segment an image into its components. This is achieved by labeling each pixel of an image with one of C known categories. Semantic segmentation can be useful for a variety of applications, including aiding autonomous vehicles navigate the world. In such an application, it is crucial that the underlying decision problem of labeling pixels be solvable in near real-time. Existing loss-calibrated approaches struggle with such real-time requirements. through this experiment, we demonstrate both real time performance and improved decisions provided by our post-hoc loss correction framework.

6.2.4.1 Experimental setup

For this experiment, we use the Camvid data set [Brostow et al., 2008a,b] which contains per-pixel labeled images captured using a camera on the dashboard of a car. An illustration of the data is provided in Figure Fig. 6.5. The version of the data set that we use contains a total of 12 class labels. Of these class labels, we assign a lower cost to false decisions that involve picking either of the pedestrian, cyclist or car class. This decision cost structure is inspired from the experiments of Cobb et al. [2018a], as the goal of an autonomous car is to avoid these obstacles for safety reasons. By assigning a lower penalty to an incorrect decision of classifying a pixel to one of these three classes, we encourage our model to be more risk averse. For our model, we use the SegNet model architecture [Badrinarayanan et al., 2017]. SegNet is an auto-encoder style model which has previously been used for semantic scene segmentation. For additional comparison, we add the loss calibrated MC dropout baseline based on the algorithm presented in Cobb et al. [2018a].

For the SGHMC implementation in this experiment, we use an initial learning rate of 0.01, a momentum of 0.5 and effective weight decay of 5×10^{-4}. We also use a cosine annealing learning rate scheduler, which decays the learning rate to 0. We run a burn-in phase of 1000 iterations, and collect total 30 parameter samples with a thinning interval of 500 iterations.

We run the distillation algorithm alongside the sampling to train the student model for a total of 15000 iterations after the burn-in. To train the student model, we use SGD with a cyclic learning rate schedule. The maximum learning rate in the schedule is 0.15, with a momentum of 0.9 and a weight decay factor of 10^{-4}. Once we obtain the student, we train our $q(.)$ model using the objective in Eq. (6.11). For training the $q(.)$ model,we use SGD with a cyclic learning rate schedule. The maximum learning rate in the schedule is 0.05, with a momentum of 0.9. Note that for running SGHMC we start with a pre-trained solution. The pre-trained solution used in this experiment is the CW-SGD solution. More details on CW-SGD solution is given later in this section. Once we obtain the student model S, we train the $q(.)$ model, using SGD with a cyclic learning rate schedule. The maximum learning rate in the schedule is 0.05, with a momentum of 0.9 for a total of 15000 iterations.

For the CW-SGD baseline, we use a cyclic learning rate schedule and train the SegNet model for 15000 iterations with a maximum learning rate of 0.05, weight decay of 10^{-4} and momentum of 0.9. The classes of importance here are the car, cyclist and pedestrian, and hence the loss weights for these classes are set to 1.4 while the rest of the loss weights are set to 1.

For the loss calibrated MC dropout baseline, we use the pretrained SGD solution, and fine-tune it with Adam optimizer using a learning rate of 10^{-4} for 1000 training iterations. The dropout strength is 0.3 and is added in the decoding phase of the model. For computing the posterior predictive distribution at train, we use 5 forward passes, and during testing, we use 30 forward passes. Additional details about this algorithm can be found in Cobb et al. [2018a].

The decision cost matrix for this experiment is shown in Table 6.3. This decision cost matrix puts less penalty on decisions involving the classes of high importance.

6.2.4.2 Results

The aim of this experiment is two-fold. First, we want to evaluate the performance of our approach for the task, and compare it with relevant baselines. Secondly, we want to assess the test-time efficiency of our approach against alternates.

For assessing performance, we use the intersection over union (IoU) ($\uparrow$) metric that is commonly used to assess semantic segmentation performance. This metric evaluates the ratio of the area of overlap and the area of union while comparing the ground truth

Table 6.3. Semantic scene segmentation. Decision cost matrix for the semantic scene segmentation experiment.

						Decision						
Cost	Sk.	Bu.	Po.	Ro.	Pa.	Tr.	Si.	Fe.	Ca.	Pe.	Cy.	Un.
Sky	0.	0.8	0.6	0.6	0.6	0.6	0.6	0.6	0.4	0.4	0.4	0.6
Building	0.8	0.	0.6	0.6	0.6	0.6	0.6	0.6	0.4	0.4	0.4	0.6
Pole	0.8	0.8	0.	0.6	0.6	0.6	0.6	0.6	0.4	0.4	0.4	0.6
Road	0.8	0.8	0.6	0.	0.6	0.6	0.6	0.6	0.4	0.4	0.4	0.6
Pavement	0.8	0.8	0.6	0.6	0.	0.6	0.6	0.6	0.4	0.4	0.4	0.6
Tree	0.8	0.8	0.6	0.6	0.6	0.	0.6	0.6	0.4	0.4	0.4	0.6
Sign	0.8	0.8	0.6	0.6	0.6	0.6	0.	0.6	0.4	0.4	0.4	0.6
Fence	0.8	0.8	0.6	0.6	0.6	0.6	0.6	0.	0.4	0.4	0.4	0.6
Car	0.8	0.8	0.6	0.6	0.6	0.6	0.6	0.6	0.	0.4	0.4	0.6
Pedestrian	0.8	0.8	0.6	0.6	0.6	0.6	0.6	0.6	0.4	0.	0.2	0.6
Cyclist	0.8	0.8	0.6	0.6	0.6	0.6	0.6	0.6	0.4	0.2	0.	0.6
Unlabelled	0.8	0.8	0.6	0.6	0.6	0.6	0.6	0.6	0.4	0.4	0.4	0.

Ground Truth/Prediction

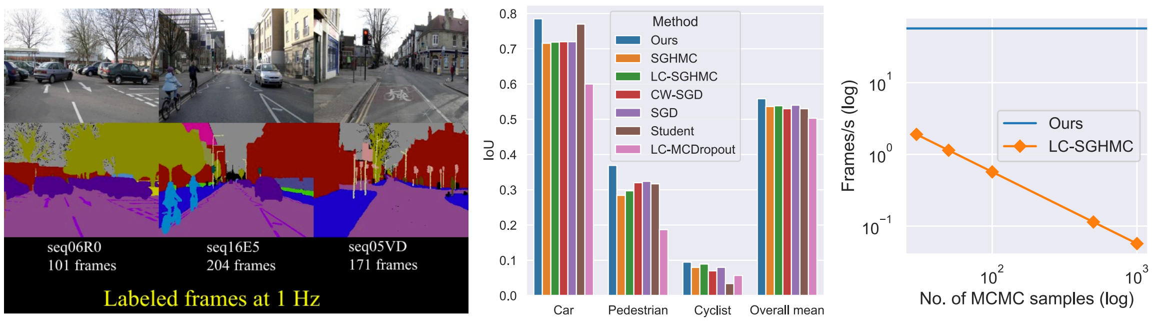


Figure 6.5. Semantic Segmentation (Left) Sample input images (top) and ground truth segmentations (bottom) from the CamVid dataset. (Middle) IoU scores achieved by different methods for classes deemed important by the cost function, as well as overall mean IoU across all classes and all images. (Right) We compare the number of frames processed per second by our method and LC-SGHMC. Owing to amortization, our approach is independent of the number of samples used to compute the Monte Carlo approximation to the posterior predictive distribution. In contrast, the number of frames processed per second by LC-SGHMC decreases dramatically (note the log scaling of the axes) with increasing number of samples.

segmentation and model output segmentation for each class. In Fig. 6.5, we present a performance comparison between our method and the baselines using the IoU metrics on test set. For the comparison, we look at each of the high utility classes separately, as well as we evaluate the overall mean IoU over all classes combined. We observe that our method performs consistently better on all the high utility classes, and thus doing a better job at capturing the preferences embedded in our cost function. Moreover, the mean IoU across all the classes indicates that our method does a better job overall for this task.

Next, we look at assessing test-time efficiency for our current task. The metric used for quantifying the test time efficiency is the number of frames that can be processed per second (FPS, $\uparrow$). For application such as autonomous driving, a slower processing pipeline can create a bottleneck when it comes to the efficacy for deployments. However, since we use a point-estimated model for our approach, it gives us inherent time savings when looking at test-time processing capabilities. In Figure 6.5, we present a comparison of no. of frames/sec that can be processed between our approach and LC-SGHMC. Each frame (image) in the data set has a resolution of 360×480. We compute the time to process an image after loading both the image and model (or model ensemble for LC-SGHMC) on a Nvidia Tesla V100 GPU. While we expect Monte Carlo based approximations to perform much slower than our point-estimated model, it is increasingly evident looking at Figure 6.5 (middle) that the number of frames/s can be prohibitively low for practical applications, for even a smaller number of MCMC samples. While the performance for MCMC methods in Figure 6.5 is computed using 30 samples, we compute the frames/s metric over larger ensemble sizes to give a sense of why Monte Carlo integration at test time can be impractical.

To summarize, there are two key findings of this experiment. First, our approach leads to improved decision-making by capturing class preferences better, and improves on the overall performance averaging across all twelve classes. Second, our method is better positioned for deployments requiring real-time performance, as at decision time it requires a single forward pass and no Monte Carlo approximations.

6.3 Conclusions

In this chapter, we introduced a novel framework for post-hoc loss calibration of Bayesian neural networks for decision-making. Through comprehensive empirical evaluations ranging from synthetic data sets to practical applications involving real world data, we have demonstrated that our approach consistently produces lower cost, higher utility decisions than competing approaches. We also demonstrated that by decoupling posterior inference from decision-making, the framework provides computational advantages at training time, and through amortization provides fast test-time decisions. Future directions include extensions to continuous decisions, more carefully exploring the effect of the choice of an inference algorithm on the quality of the correction, exploring post-hoc corrections under distribution shift, and studying the connections to generalized Bayesian inference [Bissiri et al., 2016, Knoblauch et al., 2019].

CHAPTER 7

UNCERTAINTY-AWARE OBJECT DETECTION

Bayesian deep learning methods improve the reliability of model confidence by accounting for uncertainty in parameter estimation. This is advantageous in situations where we look to deploy our models for high-stakes applications. In this chapter, we explore a large-scale practical application of these methods: object detection. Recently, a new paradigm of query-based object detection has gained popularity, such as Detection Transformer (DETR), an encoder-decoder-based object detection model, and AdaMixer. In this chapter, we focus on quantifying multiple aspects of detection uncertainty based on a deep ensembles representation for the aforementioned models [Carion et al., 2020, Lakshminarayanan et al., 2017, Gao et al., 2022].

The remainder of the chapter is structured as follows. In Section 7.1, we introduce the object detection task and briefly present related work in this area. In Section 7.2, we describe the architecture of the DETR model and the training process, in more detail. In Section 7.3, we present our evaluation principles for the task. We present our experimental analysis based on these principles in Section 7.4. We provide conclusions in Section 7.5 along with potential future directions.

7.1 Background and Related Work

In the task of object detection, the goal is to identify the set of objects present in a given scene, along with their locations and classes. The location of the object is specified using a bounding box. An illustration of the object detection task is presented in Figure 7.1.

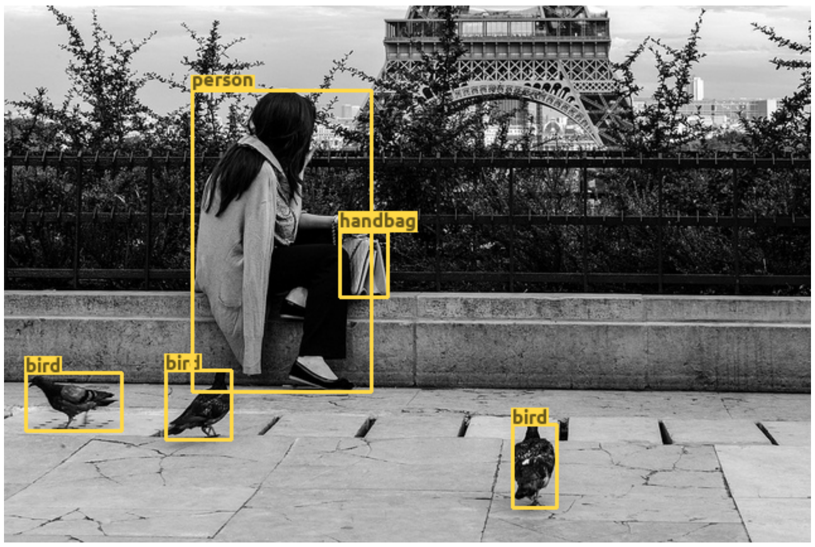


Figure 7.1. A sample image from the COCO dataset [Lin et al., 2014].

There has been much prior work that builds on convolutional neural networks for the purpose of object detection. An important distinction between the image classification task and the object detection task is that in the former only one object of interest in the scene has a specified label, while in the latter case there could be many objects of interest in the scene. As a result, in addition to identifying the classes of objects, we also need to specify the locations of the objects.

One of the earliest methods in this area uses a sliding-window approach where we generate different crops of an image using sliding windows, and for each of the windows we classify if the cropped area is part of the background or if it contains an object of interest. If it contains an object of interest, we also predict the class of the object. This approach can be problematic as it requires generating a very large number of candidate bounding boxes.

There have been approaches that aim to tackle this issue by generating a smaller set of candidate bounding boxes based on classical computer vision techniques [Alexe et al., 2012, Uijlings et al., 2013, Cheng et al., 2014, Zitnick and Dollár, 2014]. R-CNN [Girshick et al., 2014] (Region-CNN) is the earliest work in this area that combines region proposal with CNNs for object detection. R-CNN contains a classification loss to learn the object class correctly, as well as a regression loss that aims to improve bounding box prediction, in addition to the region proposal network. However, an important shortcoming of this method is that it requires multiple forward passes for the regions of interest through a deep CNN to make predictions, which increases the prediction latency.

To tackle this, Fast R-CNNs were proposed [Girshick, 2015], which generate the feature maps for the image once, as well as project the regions of interest onto the feature maps. As a result, we can reuse the feature maps, which reduces the computational complexity at inference by a large factor. The main bottleneck in Fast R-CNNs has been the region proposal methods, which, as we noted earlier, work based on classical computer vision techniques. To solve this further, Faster R-CNNs were proposed, which includes a region proposal network (RPN). The RPN sits on top of the feature maps generated by the deep convolutional neural networks and generates its own set of region proposals. Aside from this, the entire framework is the same as Fast R-CNNs [Ren et al., 2016]. This leads to a significant reduction in prediction runtime, making it an attractive method for practical applications.

A common aspect of the approaches discussed above is that they all require a proposal of initial regions to identify objects and make predictions. Another family of approaches involve object detection without region proposals [Redmon et al., 2016, Liu et al., 2016]. These approaches involve dividing the input image into a dense grid, and for each cell in the grid, predict for multiple bounding boxes (the number of which is predetermined) the relative location and size of the bounding bbox. Given that these models do not require generating multiple regions and computing output for each of them, these methods are much faster than region proposal-based methods. Note that all these methods also involving post-processing steps to clean up the multiple bounding boxes from the prediction phase.

With the advent of vision transformers, there has also been prior work in this area which looks at adapting vision transformers for end-to-end object detection. DETR (Detection Transformer) by Carion et al. [2020] is the most prominent piece of work in this area. This model consists of a standard transformer-based encoder-decoder framework where the entire image is passed through the encoder, and the hidden representation from the encoder is then fed to the decoder. Next, we predetermine the maximum number of bounding boxes to generate for our image, and then pass a sequence of learned position embeddings (also known as object queries) to the decoder, and get the output for the bounding box corresponding to each of these position embeddings. Finally, there is also additional work that looks at using transformer-style models for end-to-end object detection. For example, Zhu et al. [2021] proposes *Deformable DETR*, which includes a new component termed deformable attention in the encoder module. This helps speed up the training process of DETR and also does a better job at identifying smaller objects. Song et al. [2021] introduces a novel integration of vision transformers (ViT) and the DETR framework to achieve a much better performance-latency trade-off than the original DETR framework.

There has been some prior work on approximate Bayesian inference for deep CNN-based object detection models. Much of this work looks at estimating uncertainty in the prediction of bounding-box properties such as location and size. Kendall and Gal [2017] was one of the earliest works in this space that proposed using log-likelihood loss to estimate heteroscedastic aleatoric uncertainty for the bounding-box regression. Le et al. [2018] build upon this to have the neural network model explicitly produce an estimated output variance like in heteroscedastic regression. This requires a single forward pass through the neural network to estimate the output variance. Miller et al.

[2018] use MC Dropout and then compute the mean and variance in the context of bounding-box regression by computing statistics over spatially correlated outputs. The BayesOD approach [Harakeh et al., 2020] builds upon Le et al. [2018] to get rid of the non-maximal suppression with Bayesian inference and provides a multivariate extension instead of the diagonal covariance of the standard log-likelihood for bounding box regression.

The widely used metric for evaluating object detection models is the mean average precision (mAP) score, computed by averaging the precision scores over a range of matching IoU thresholds. However, the mAP score only evaluates the accuracy of the hard predictions but doesn't account for the uncertainty in model predictions. Previous work by Hall et al. [2020] proposed *Probability-based Detection Quality (PDQ)*, a metric to evaluate the probabilistic aspect of the outputs. PDQ consists of two components: spatial quality, and label quality. However, there are certain issues with the PDQ evaluation framework. First, the label quality is measured as the probability of the correct class, with the unmatched ground truth and predicted bounding boxes contributing a score of 0. This is problematic because, as we highlight later in Section 7.3, a more reasonable way to assess the class uncertainty on unmatched predicted bounding boxes is to consider the NLL of the background class prediction. Second, while looking at spatial quality, the metric looks at two components: foreground loss and background loss. Foreground loss is the negative log-likelihood of the pixels in the ground truth bounding box under the predicted distribution that assigns a probability value on whether a given pixel in the scene is a part of the detection. Background loss penalizes pixels by using the complement of the predicted distribution, by computing the NLL of the pixels present in the predicted bounding box, but not in the ground-truth bounding box. This is again problematic, as it composes location uncertainty with evaluating bounding box coverage and precision (see Section 7.3.3 for the definition of bounding box coverage and precision) in a way that makes it harder to decompose them. Finally, while matching ground truth bounding boxes to the predicted bounding boxes, the authors propose computing PDQ scores for every combination of the predicted bounding boxes and ground truth bounding boxes. This can lead to leakage of label information at the time of matching, which is not desirable.

In this chapter, we focus on the original DETR framework. To the best of our knowledge, the DETR framework has not been explored in the context of uncertainty estimation in the literature.

7.2 Uncertainty Estimation in Query-Based Detectors

In this section, we first describe the specific query-based detectors we use in more detail. We then explain our proposed approach to merge the output of an ensemble of these detectors. Finally, we describe how we quantify different kinds of uncertainty derived from the ensemble.

7.2.1 Detection Transformer and AdaMixer

At the most abstract level, query-based detectors can be thought of as learning a set of query embeddings in a latent embedding space that are decoded into bounding box predictions. A decoder function is then learned, which takes as input the image features from a CNN backbone (or encoder, for e.g. in the case of DETR) and the query embeddings. The output of the decoder is a new set of transformed query embeddings of the same shape. These can then be transformed into class and bounding box predictions using MLP output heads. As we described in the previous section, the DETR model is an attention-based encoder-decoder architecture with a pre-trained convolutional neural network-based backbone.

In the case of DETR and AdaMixer, a set of 100 256-dimensional query embedding vectors is used. Furthermore, both models use ResNet50 as the backbone [He et al., 2016b]. In the case of DETR, the decoder is a transformer that computes cross-attention between image features and query embeddings. AdaMixer replaces the transformer with an MLP mixer model. This decoder uses position-aware adaptive mixing between the query embeddings and image features. This has been shown to result in faster convergence and better performance than DETR.

For each bounding box at index i in the train dataset, the ground truth can be denoted as $y_i = (c_i, b_i)$, where c_i denotes the class, and $b_i \in [0,1]^4$ denotes the x, y coordinates of the center of the box, as well as the height and width relative to the image. Finally, the framework uses the *Hungarian* loss function, which combines the classification loss and box loss, as shown below.

$$\mathcal{L}_{\text{Hungarian}}(y, \hat{y}) = -\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbb{1}_{\{c_i \neq \text{``background''}\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}}(i)) \tag{7.1}$$

Here, $\hat{\sigma}(i)$ denotes the permutation of the predictions that minimizes the matching loss between predictions and the ground truth. Furthermore, $\hat{p}_{\hat{\sigma}(i)}(c_i)$ denotes the predicted probability for class c_i under the $\hat{\sigma}(i)$ permutation, and $\hat{b}_{\hat{\sigma}}(i)$ denotes the predicted bounding box coordinates under the $\hat{\sigma}(i)$. It's important to note that this matching computation is only required during training. $\mathcal{L}_{\text{box}}$ is constructed using two components: the IoU loss ($\mathcal{L}_{\text{iou}}$) and the L1 regression loss. The IoU loss ensures there is high overlap between the ground truth bounding box and the predicted bounding box, while the L1 regression loss ensures that the location specifications predicted for the bounding box are closer to the ground truth.

$$\mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}}(i)) = \lambda_{\text{iou}} \mathcal{L}_{\text{iou}}(b_i, \hat{b}_{\hat{\sigma}}(i)) + \lambda_{\text{L1}} \left\| b_i - \hat{b}_{\sigma(i)} \right\|_1 \tag{7.2}$$

Here, λ_{iou} and λ_{L1} denote the hyperparameters for weighing the two components of the loss function.

7.2.2 Merging Bounding Boxes

Unfreezing the decoder parameters leads to the reinitialization and retraining of the object queries. This results in a shuffling of the order of the bounding-box predictions between different members of the ensemble. To merge the bounding box predictions emerging from different members of the ensemble, we resort to the technique of clustering. We use the Basic Sequential Algorithmic Scheme (BSAS) [Theodoridis and Koutroumbas, 2006, Miller et al., 2019] method to group the bounding boxes. For a given input, consider the set of outputs of each of the members of the ensemble as $D_k = \cup\{(b_{ik}, s_{ik})\}_{i=1}^{N}$, where i denotes the index of the output bounding box, k denotes the index of the model in the ensemble, b denotes the predicted output corresponding to the location of the bounding box, and s denotes the predicted class distribution for the bounding box. Furthermore, let $\mathcal{O}_m$ denote the m^{th} cluster. Under the algorithm, we sequentially merge detections into clusters if the maximum IoU between detection D_i and any existing $\mathcal{O}_m$ is greater than or equal to a set threshold ω. Otherwise, the detection is assigned as a cluster of its own. Once the clusters are formed, for computing the posterior predictive distribution, we use the standard Monte Carlo approximation as shown below.

$$\hat{s}_i = \frac{1}{K}\sum_{k=1}^{K} s_k, \text{ where } K = |\mathcal{O}_i| \tag{7.3}$$

For clustering, we use the implementation provided by Miller et al. [2021], and its worst-case runtime complexity is $\mathcal{O}((KN_D)^2)$, where K denotes the ensemble size, and N_D denotes the maximum number of detections for a model in the ensemble. Next, for merging the bounding box location predictions, there are several options. First, we could simply compute the "average" bounding box, formed by taking the average of the top-left and bottom-right coordinates. In fact, the average merging strategy has been used in the past to merge bounding boxes [Miller et al., 2021]. In addition to the average bounding box, we could also generate a "maximal" bounding box or a "minimal" bounding box. A maximal bounding box, by definition, is the tightest bounding box that encapsulates all the bounding boxes within a cluster. On the other hand, a minimal bounding box is the intersection of all the bounding boxes within a cluster. Consider the bounding box $b_j = (x_{1j}, y_{1j}, x_{2j}, y_{2j})$ in a cluster $\mathcal{O}_i$, where (x_{1j}, y_{1j}) indicate the coordinates of the top-left corner, and (x_{2j}, y_{2j}) indicate the coordinates of the bottom-right corner, then the merged bounding is obtained by :

$$\hat{b}_i^{\text{avg}} = \frac{1}{J}\sum_{j=1}^{J} b_j, \text{ where } J = |\mathcal{O}_i| \text{ (average bounding box)} \tag{7.4}$$

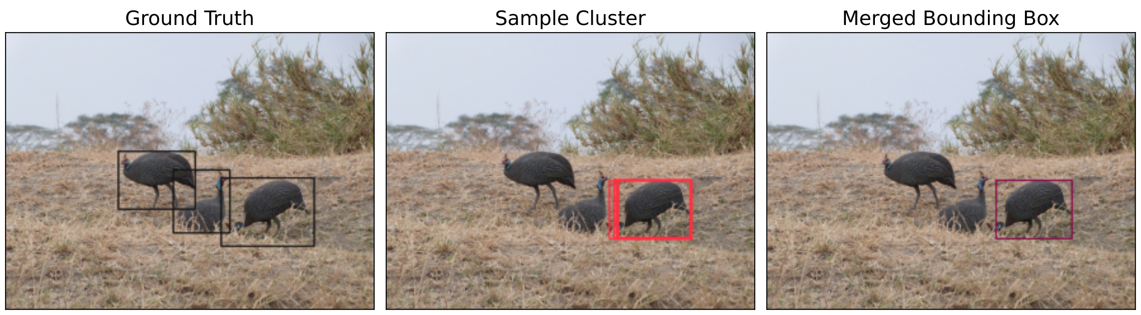
$$\hat{b}_i^{\text{max}} = (\min(\mathbf{x_1}), \min(\mathbf{y_1}), \max(\mathbf{x_2}), \max(\mathbf{y_2})) \text{ (maximal bounding box)} \tag{7.5}$$

$$\hat{b}_i^{\text{min}} = (\max(\mathbf{x_1}), \max(\mathbf{y_1}), \min(\mathbf{x_2}), \min(\mathbf{y_2})) \text{ (minimal bounding box)} \tag{7.6}$$

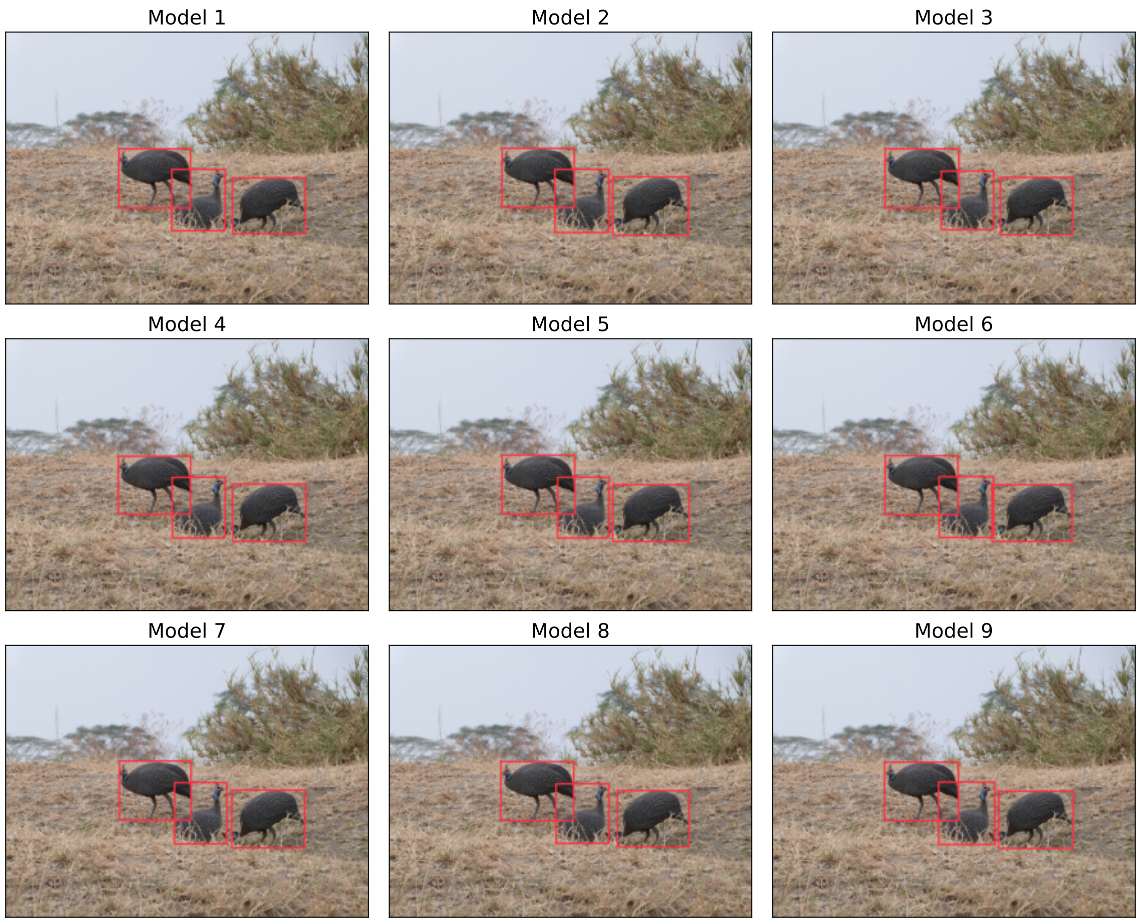
In the above set of equations, $\mathbf{x_1} = [x_{11}, x_{12}, ..., x_{1J}]$, $\mathbf{y_1} = [y_{11}, y_{12}, ..., y_{1J}]$, $\mathbf{x_2} = [x_{21}, x_{22}, ..., x_{2J}]$, and $\mathbf{y_2} = [y_{21}, y_{22}, ..., y_{2J}]$. An illustration depicting individual bounding box predictions of different members of the ensemble and merging strategy has been given in Figure 7.2. Note that the clustering method involves representing the bounding box by its top-left and bottom-right coordinates, instead of the representation using center coordinates and bounding box dimensions that DETR typically outputs.

7.3 Evaluation Principles and Methods

In this section, we describe the evaluation principles of interest for our work. To evaluate our uncertainty-aware object detection models, we focus on four distinct aspects of our predictions: 1) location uncertainty, 2) classification performance (for class uncertainty), 3) coverage and precision of predicted bounding boxes, and



(a) (Left) Image with ground truth bounding boxes. (Middle) Clustered bounding boxes from the full ensemble for one object. (Right) Average bounding box post merging for one object.



(b) Bounding box predictions for individual models in the full ensemble

Figure 7.2. An illustration of the predictions of the individual members of the full ensemble, the clustered bounding boxes for a sample object, and the final average bounding box. For more context, we also present the ground truth bounding boxes.

4) objectness detection. In the following subsections, we describe each of these performance aspects in more detail.

7.3.1 Location Uncertainty

The first aspect of predictive uncertainty that we evaluate is the uncertainty in the location of the object. Specifically, we look at the uncertainty in the center location of the output bounding box. To evaluate the location uncertainty, we compute the negative log likelihood of the true location of the center $(x_{\text{center}}, y_{\text{center}})$ under a Gaussian distribution $\mathcal{N}(\hat{\mu}, \hat{\sigma}^2)$, where $\hat{\mu}$ is the predicted center location, that is,

$$\hat{\mu} = (\hat{x}_{\text{center}}, \hat{y}_{\text{center}}) \tag{7.7}$$

$$\hat{\sigma}^2 = \left(\frac{1}{N} \sum_{n=1}^{N} (\hat{x}_{\text{n,center}} - x_{\text{n,center}})^2, \frac{1}{N} \sum_{n=1}^{N} (\hat{y}_{\text{n,center}} - y_{\text{n,center}})^2 \right) \tag{7.8}$$

In the above set of equations, $(x_{\text{n,center}}, y_{\text{n,center}}) \in \mathcal{D}_{\text{boxes,train}}$, where $\mathcal{D}_{\text{boxes,train}}$ denotes the collection of the bounding boxes from the entire training set. The variance computation is equivalent to computing the average of the squared residuals of the model predictions. A prerequisite to computing the variance is that we first need to match the ground-truth bounding boxes with the predicted bounding boxes on the training data. Predicted bounding boxes are assigned to ground truth bounding boxes by solving the minimum weight matching problem, where each matching has a weight of (1 - IoU). Furthermore, we set a threshold on minimum IoU for the matching between the predicted bounding box and the ground truth bounding box to be valid during evaluation. An important point to note here is that we can only compute the variance using the matched bounding boxes on the train data. Although the equations mentioned earlier look at the predictions from a single model, this can be extended in a straightforward way for ensembles by using a Gaussian mixture model likelihood, where the mixture indicates the cluster forming the detections, and thus each member of the mixture is an individual detection. For every member of the mixture, the mean parameter is the center of the detection, and the variance is the average of the variances across all members of the ensemble (each computed using the earlier given above).

7.3.2 Classification performance

Along with each bounding box, the DETR model outputs a $(K + 1)$ dimensional probability distribution, where K denotes the number of classes and the final dimension is reserved for the "background" class. If the bounding box has maximum probability on the background class, it is filtered out during evaluation. Before we can compute any metrics for the classification performance, we need to specify the assignment procedure that assigns predicted bounding boxes to the ground truth bounding boxes.

We use the assignment described in the previous subsection for assigning the predicted bounding boxes to ground truth bounding boxes. For the predicted bounding boxes that do not get assigned to any ground truth bounding box, we set their respective ground truth class to background. On the other hand, if after assignment there exist ground-truth bounding boxes that do not have any assigned predicted bounding boxes, we assert a uniform distribution over classes as the predicted distribution. This serves as a penalty for bounding box targets that are missed.

Once this assignment is complete, we evaluate the negative log-likelihood (NLL), expected calibration error (ECE), and accuracy for this multi-class classification task. NLL and ECE are especially useful as they account for the probabilistic nature of the output. For detailed definitions of these metrics, refer to Chapter 3.

7.3.3 Bounding box coverage and precision

Aside from location uncertainty and class uncertainty, it is also important to understand how well the predicted bounding boxes capture the area occupied by the underlying objects in the scene. To quantify the detector's ability to correctly localize objects in the scene, we look at bounding box coverage and precision. Specifically, coverage measures the proportion of the ground truth bounding box area that is covered by the predicted bounding box, and precision refers to the proportion of the predicted bounding box area that belongs to the ground truth bounding box. It is desirable to have our bounding box predictions cover as much area of the ground-truth bounding box as possible, resulting in a high coverage value. However, it is possible to achieve this by simply predicting larger bounding boxes, which unnecessarily widens the area of focus required for the downstream applications that utilize these bounding box predictions. As a result, it is equally important to look at the precision of our predictions to understand how much excess area is predicted. An illustration of the coverage and precision metrics is given in Figure 7.3.

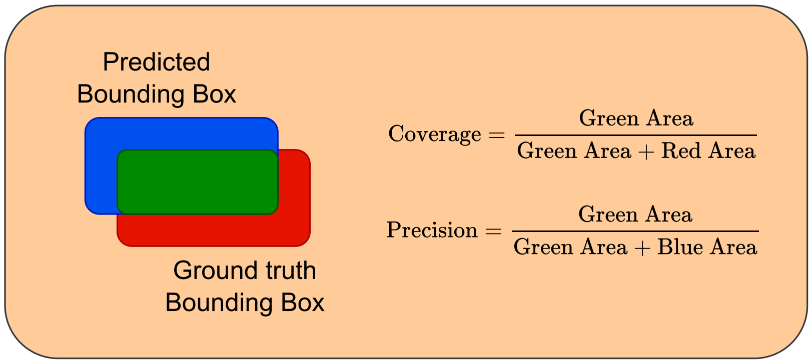

Predicted Bounding Box
Ground truth Bounding Box
Coverage = Green Area / (Green Area + Red Area)
Precision = Green Area / (Green Area + Blue Area)

Figure 7.3. An illustration of the coverage and precision metrics. An analogy can be drawn to the recall and precision metrics in the classification setting by considering the green area as the "true positive" area, the blue area as the "false positive" area, and the red area as the "false negative" area. As evident from the illustration, the denominator of the coverage metric is the area of the ground truth bounding box, and the denominator in the area of precision metric is the area of the predicted bounding box.

For matching ground truth and predicted bounding boxes, we follow the same methodology as described in the previous subsection. For each matching combination of bounding boxes, we compute the coverage and precision score and add it to our set of scores. Every unmatched ground truth bounding box, or an unmatched predicted bounding box, adds a score of 0 to our score array. This penalizes the cases where the predicted bounding boxes do not match with the ground truth bounding boxes, or the case where we predict additional bounding boxes that aren't required. It is important to note here that coverage and precision metrics are not explicitly evaluating any specific aspect of uncertainty, as it only focuses on the final bounding box outputs. Nonetheless, it is still an important aspect to evaluate the models due to the reasons described earlier in this subsection.

7.3.4 Objectness Uncertainty

In safety-critical applications of object detection models, it is often important to understand how well the detector performs when it comes to identifying the presence of "any" object in an area of interest in the scene. For example, in the case of autonomous driving, it is often sufficient to identify that there's an object in a given area in the scene for the car to avoid it, without identifying the correct class for it.

As we described earlier, among other object categories, DETR also outputs the probability that the bounding box belongs to the background class. This information can be used to predict whether the bounding box belongs to any actual object or to the background. This is a binary classification problem setup, where the probability of a bounding box belonging to an actual object is $p(\text{"Objectness"}|\cdot) = 1 - p(\text{"Background"}|\cdot)$. Next, we assign this objectness probability to every pixel belonging to the predicted bounding box. In the case where a pixel belongs to multiple bounding boxes, we take the maximum of the objectness probabilities across the multiple bounding boxes, to reflect the model's highest objectness confidence value. Generating the ground truth objectness map is very straightforward. For every pixel in the image, if it belongs to even a single ground truth bounding box, it gets labeled as belonging to an object and background otherwise.

Next, we downsample the prediction and ground-truth grids using max pooling, as processing numerous images with a high number of pixels in each of them becomes computationally prohibitive. For each image, we obtain the objectness probability and ground-truth vectors by flattening the grids, which are then finally concatenated to compute performance metrics on the objectness classification problem. The final concatenation step is important because different images in the dataset can have different sizes, and thus it is important not to bias the performance computation by assigning the same weight to metrics from different image sizes.

7.4 Experiments

In this section, we present our experimental framework and our experimental results. First, we begin by describing the experimental protocols involved, followed by different experiments and results, each looking at a specific aspect of uncertainty estimation, as described in the previous section.

7.4.1 Experimental Protocols

For our empirical evaluations, we focus on the Detection Transformer Model (DETR) and the AdaMixer model with the COCO 2017 dataset [Lin et al., 2014]. COCO 2017 is a widely used large-scale dataset for evaluating object detection models, containing 80 object categories. The COCO training dataset contains 118,287 images and a total of 860,001 bounding boxes. The COCO validation dataset contains 5,000 images, and a total of 36,781 bounding boxes. For uncertainty estimation in the models, we generate two variants of deep ensembles: a) the full ensemble and b) the decoder ensemble. In full ensemble, we train the entire model from scratch on the training data after random initialization, while in decoder ensemble, we freeze the backbone or backbone and encoder to the pre-trained model parameters, and only unfreeze and retrain the parameters corresponding to the decoder and output heads after random initialization. Each of these ensembles has 9 models, unless otherwise specified. During merging, we remove clusters that have fewer than 3 detections [Miller et al., 2021]. To train our models, we use the PyTorch-based MMDetection library [Chen et al., 2019, Paszke et al., 2019].

7.4.2 Experiment 1: Evaluating Location Uncertainty

In the first experiment, we evaluate the location uncertainty aspect of the different methods. We use the evaluation procedure reported in Section 7.3.1, and set the matching IoU threshold to 0.5. We also note that the location uncertainty is only computed for the matched bounding boxes, as setting penalties for unmatched bounding boxes is not straightforward. Furthermore, given that the dataset consists of images of different sizes, we transform the coordinates to normalized coordinates ranging between 0 and 1. We present the negative log-likelihood results for the center location prediction in Table 7.1.

Table 7.1. Location uncertainty results.

	NLL (↓)
Base DETR	-5.68
Base AdaMixer	-5.59
DETR Decoder Ensemble	-5.88
DETR Full Ensemble	-6.03
AdaMixer Full Ensemble	-5.91
AdaMixer Decoder Ensemble	-5.87

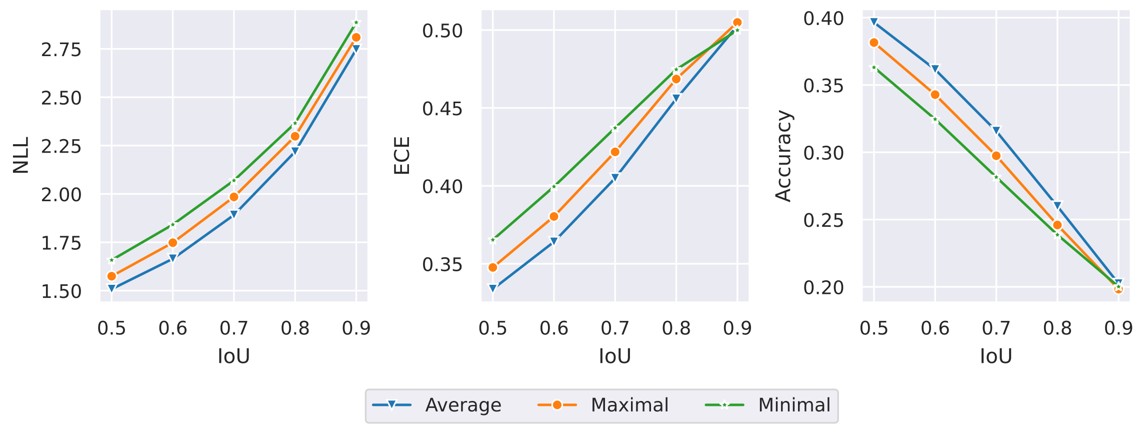
While aggregating the bounding boxes for ensemble-based methods, we use the average merging strategy. From the results presented in Table 7.1, we observe that while the full ensemble has the lowest negative log-likelihood of the methods presented for each model architecture, the performance improvement is not very significant. While comparing against models and methods, DETR full ensemble shows the best performance. Next, we turn our focus towards evaluating the class uncertainty in the model predictions. Next, we turn our focus towards evaluating the class uncertainty in the model predictions.

7.4.3 Experiment 2: Evaluating Classification Performance

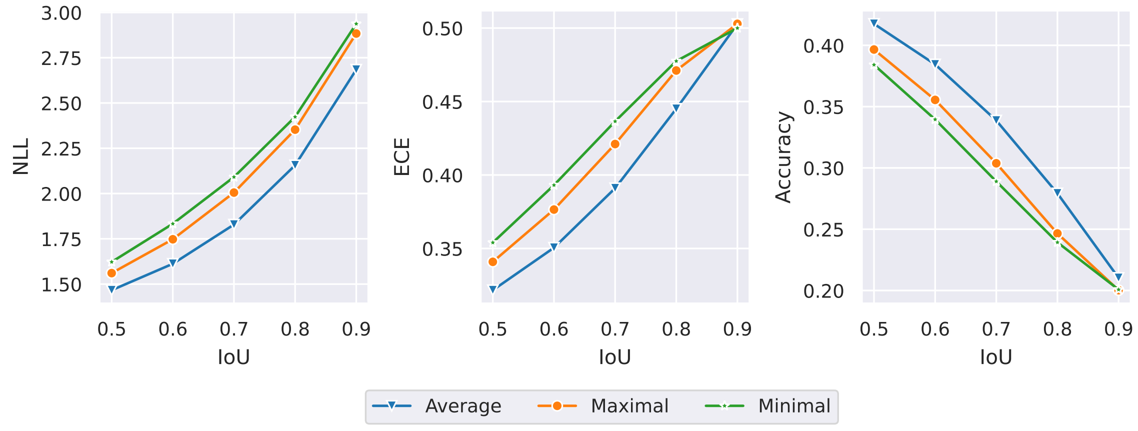
In this experiment, we evaluate and compare the classification performance across different methods and merging strategies. First, we begin by analyzing the classification performance of different merging strategies highlighted in the previous section. We present the results for different merging strategies in Figures 7.4 - 7.9.

A key difference between the results shown in Figures [7.4,7.6] and [7.5,7.7] is that the latter involves computing the metrics on only the matched bounding boxes, while the former involves computing the metrics on all the data, thus including the penalty for unmatched bounding boxes. As we set stricter matching IoU thresholds, the number of matched bounding boxes is reduced, resulting in a larger number of unmatched ground-truth and predicted bounding boxes. In this situation, we can expect the penalty terms for unmatched bounding boxes to dominate performance, and this is responsible for reducing the performance gap between methods. We also present the fraction of ground-truth bounding boxes that match in Figures 7.8 and 7.9.

Looking at the results for the entire validation set, we observe that averaging the bounding boxes in a cluster for the ensemble-based methods results in better performance across all performance metrics. However, when we look at computing the classification performance metrics on matched bounding boxes, the minimal merging strategy seems to outperform the rest in majority of instances. A potential reason for this could be that the minimal bounding boxes produced are more conservative and thus result in fewer, but more confident matches related to the bounding box area. In fact, we see in Figure 7.8 and 7.9 that the minimal strategy leads to a lower fraction of ground truth bounding boxes being matched. From a practical standpoint, this can be undesirable, as it can amount to a significant number of objects being undetected.

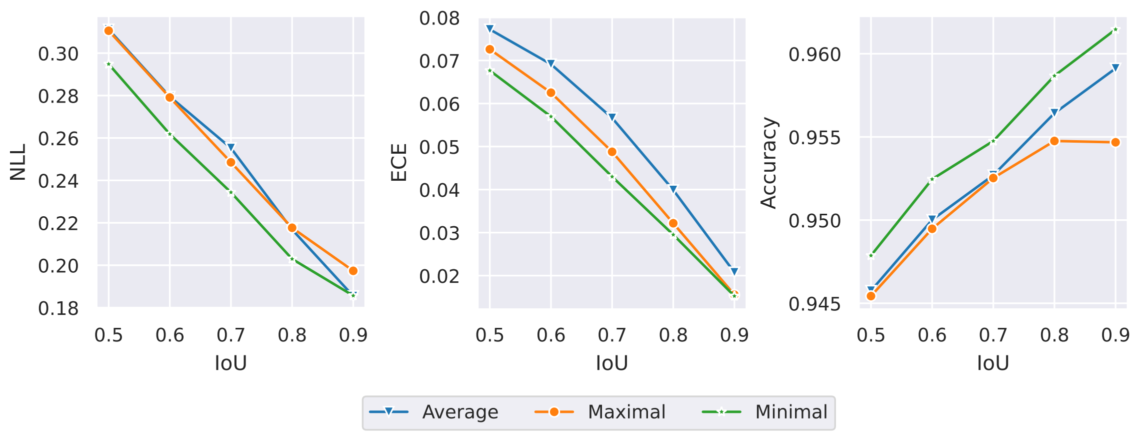


(a) Decoder Ensemble

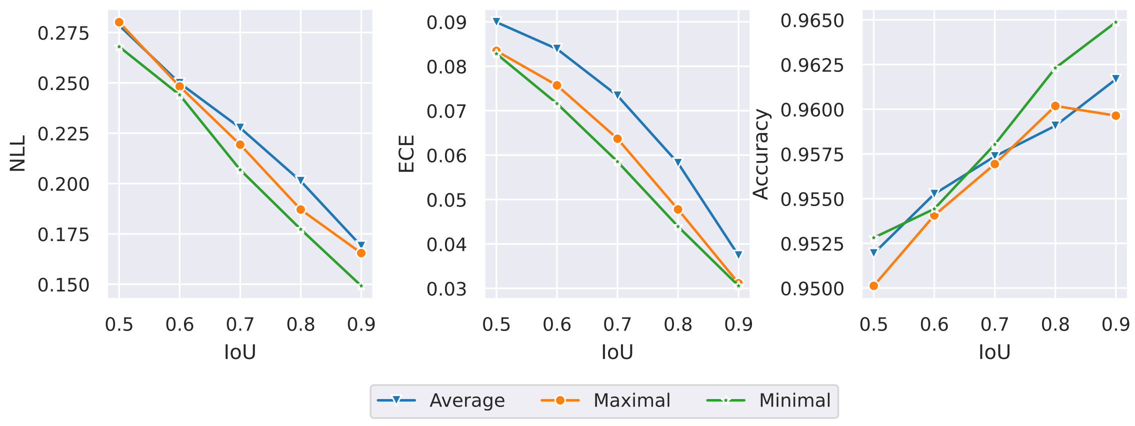


(b) Full Ensemble

Figure 7.4. A comparison of different merging strategies for decoder ensemble and full ensemble on the entire validation set for DETR model. We present negative log-likelihood (NLL, ↓), expected calibration error (ECE, ↓), and accuracy (↑) for different matching IoU thresholds.

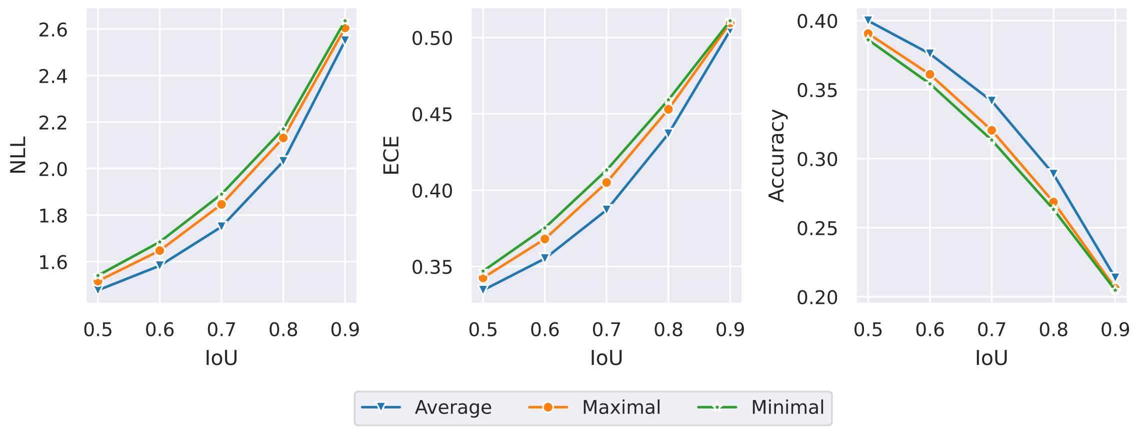


(a) Decoder Ensemble

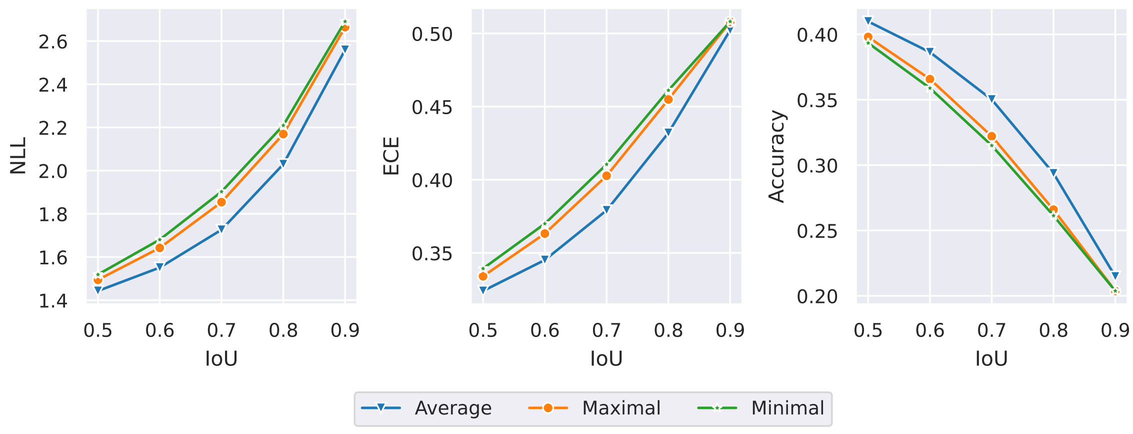


(b) Full Ensemble

Figure 7.5. A comparison of different merging strategies for decoder ensemble and full ensemble on only the matched bounding boxes for DETR model. We present negative log-likelihood (NLL, ↓), expected calibration error (ECE, ↓), and accuracy (↑) for different matching IoU thresholds.

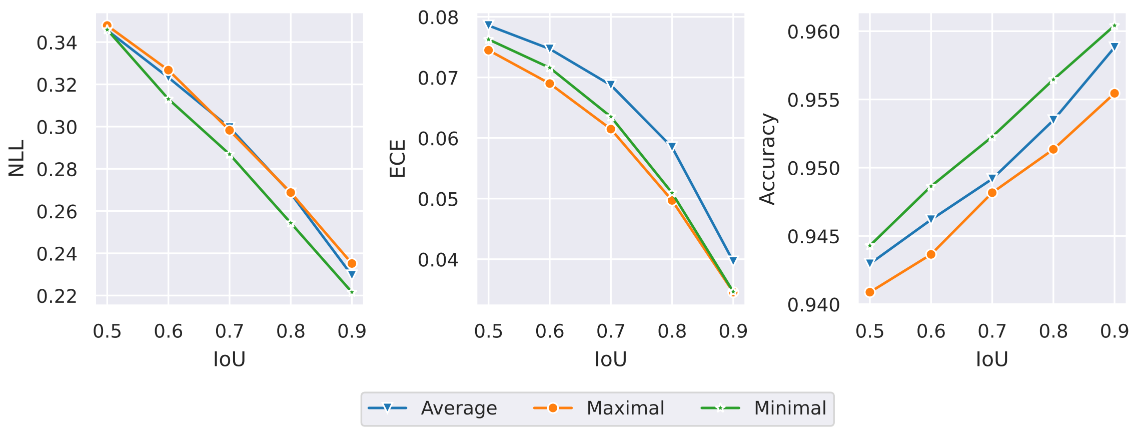


(a) Decoder Ensemble

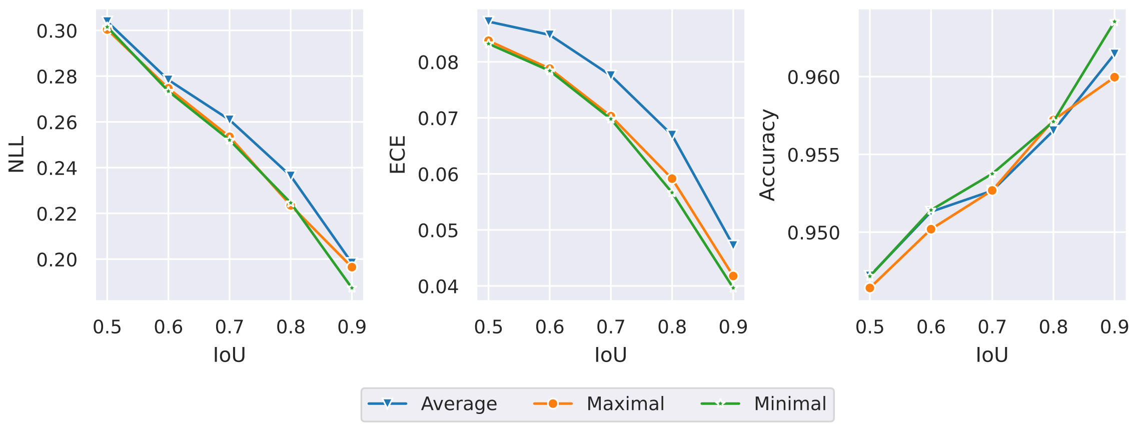


(b) Full Ensemble

Figure 7.6. A comparison of different merging strategies for decoder ensemble and full ensemble on the entire validation set for AdaMixer model. We present negative log-likelihood (NLL, ↓), expected calibration error (ECE, ↓), and accuracy (↑) for different matching IoU thresholds.

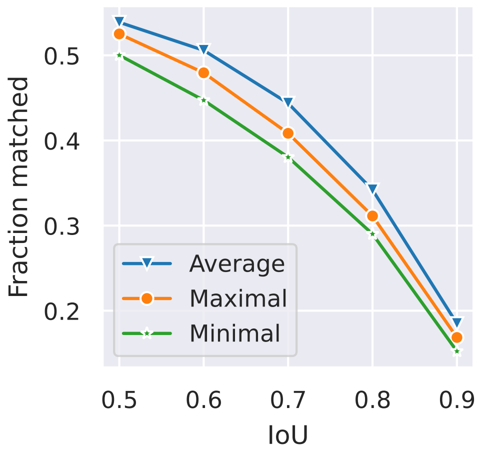


(a) Decoder Ensemble

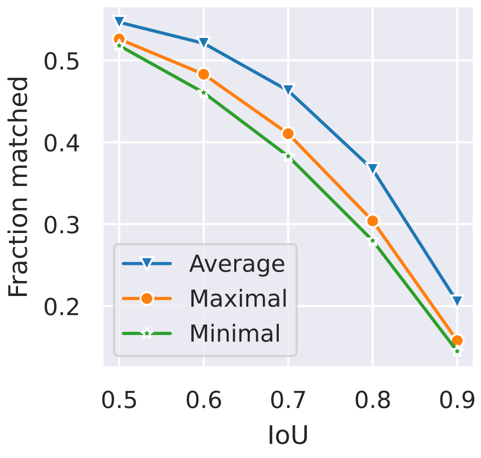


(b) Full Ensemble

Figure 7.7. A comparison of different merging strategies for decoder ensemble and full ensemble on only the matched bounding boxes for AdaMixer model. We present negative log-likelihood (NLL, ↓), expected calibration error (ECE, ↓), and accuracy (↑) for different matching IoU thresholds.

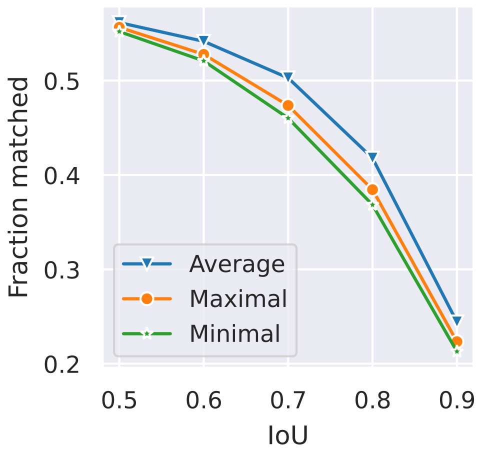


(a) Decoder Ensemble

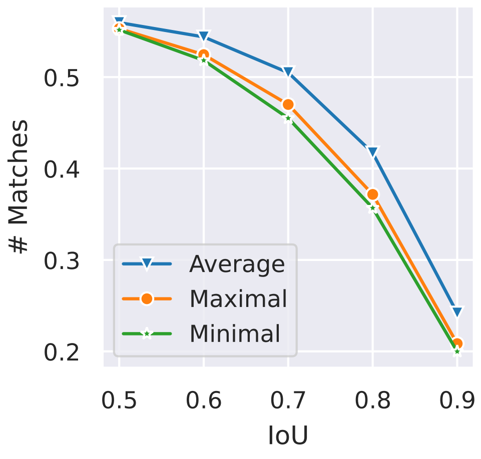


(b) Full Ensemble

Figure 7.8. Fraction of ground truth bounding boxes matched for different merging strategies across different matching IoU thresholds for DETR model.

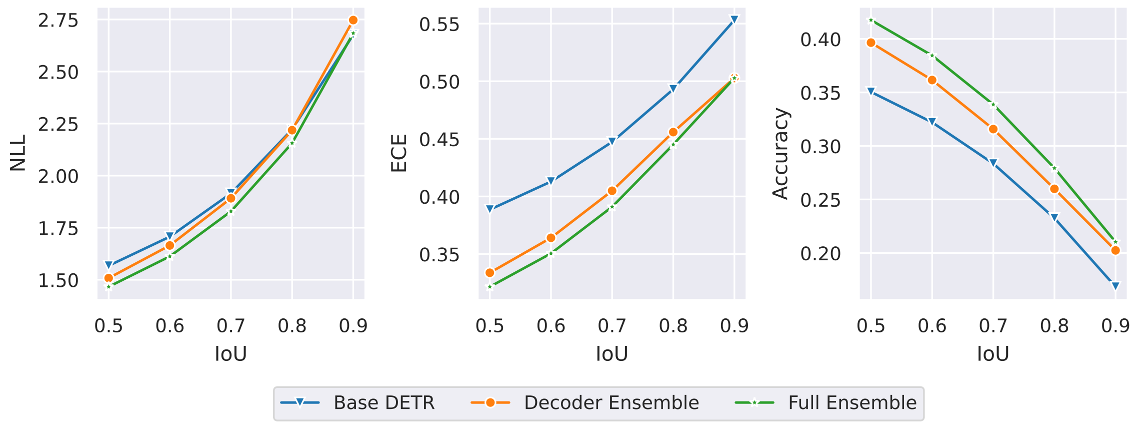


(a) Decoder Ensemble

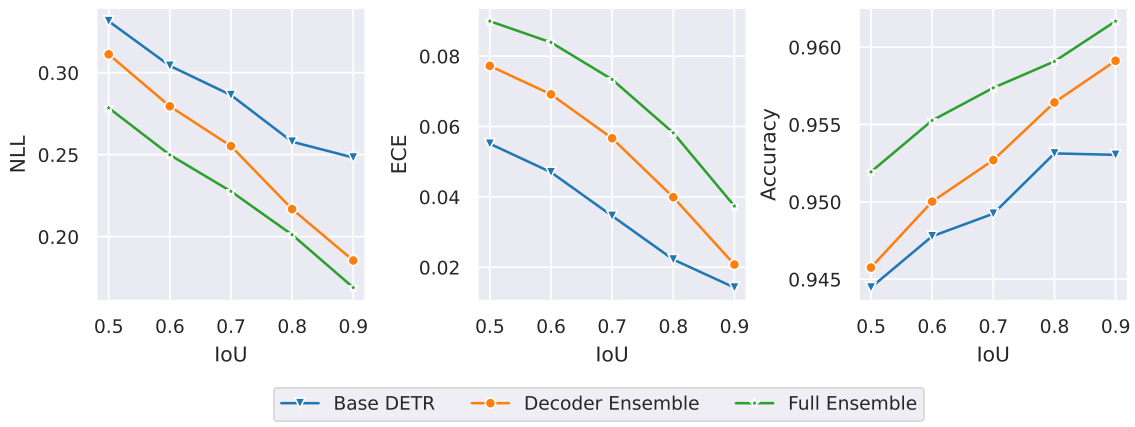


(b) Full Ensemble

Figure 7.9. Fraction of ground truth bounding boxes matched for different merging strategies across different matching IoU thresholds for AdaMixer model.

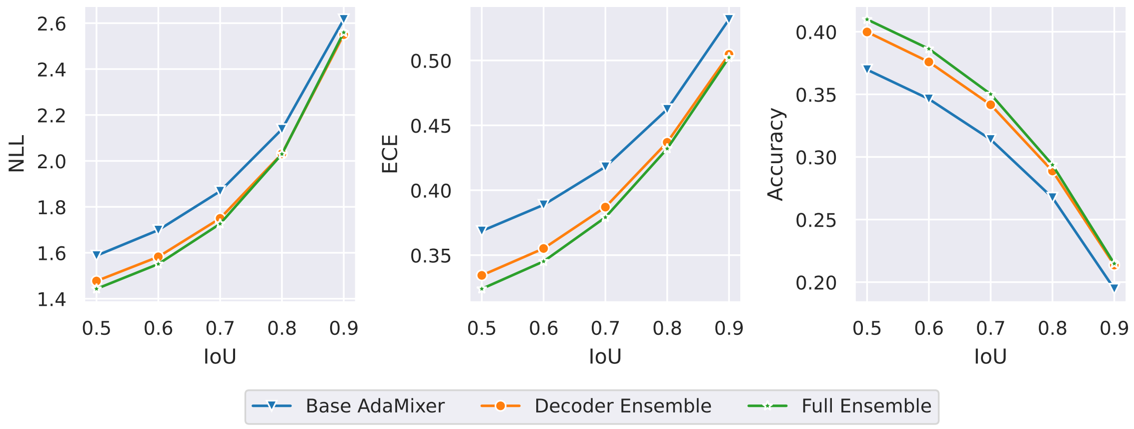


(a) All data

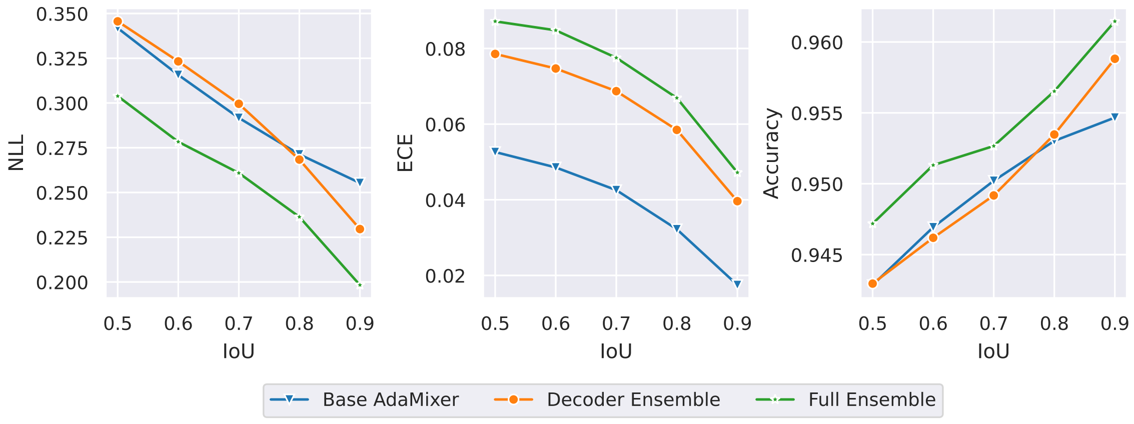


(b) Matched boxes only

Figure 7.10. A comparison of classification performance metrics across different methods for DETR model. We present negative log-likelihood (NLL, ↓), expected calibration error (ECE, ↓), and accuracy (↑) for different matching IoU thresholds. For deep ensembles, we use the average bounding box merging strategy.



(a) All data



(b) Matched boxes only

Figure 7.11. A comparison of classification performance metrics across different methods for AdaMixer model. We present negative log-likelihood (NLL, ↓), expected calibration error (ECE, ↓), and accuracy (↑) for different matching IoU thresholds. For deep ensembles, we use the average bounding box merging strategy.

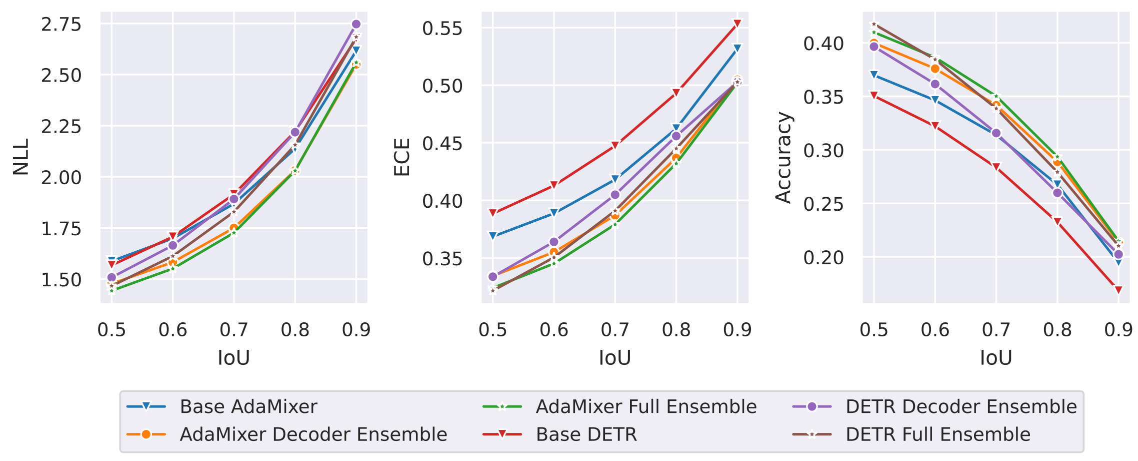
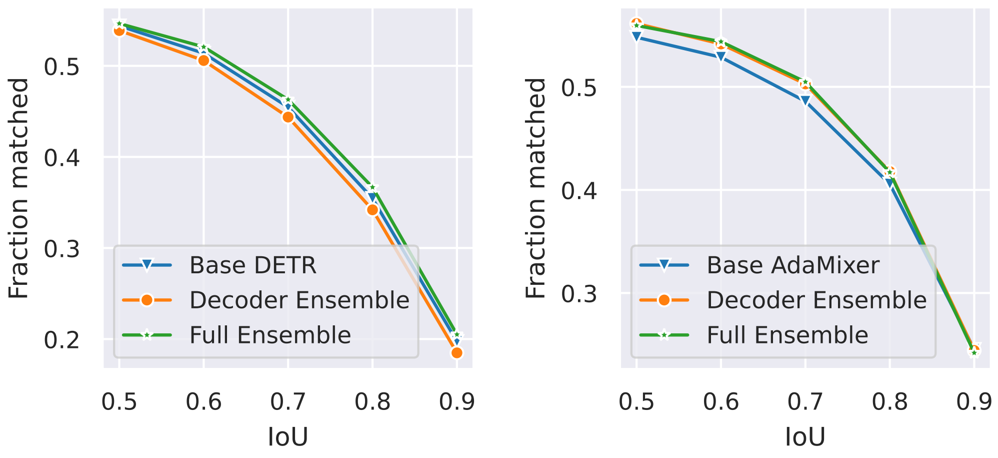


Figure 7.12. A comparison of classification performance metrics across different methods for different models on the entire validation set. We present negative log-likelihood (NLL, ↓), expected calibration error (ECE, ↓), and accuracy (↑) for different matching IoU thresholds. For deep ensembles, we use the average bounding box merging strategy.



(a) DETR

(b) AdaMixer

Figure 7.13. Fraction of ground truth bounding boxes matched for different methods across different matching IoU thresholds.

Next, in Figures 7.10 & 7.11, we present the results comparing classification performance across different methods. We focus on the average merging strategy for aggregating bounding boxes for deep ensembles as it results in better overall performance when we consider the entire validation dataset, as well as leads to a higher fraction of the ground truth bounding boxes matched. We observe that deep ensembles help us improve over the performance of a single model across all three performance metrics we measure for both models. However, especially for NLL, the performance gap closes between a single model and the ensemble as we set a stricter matching IoU threshold. This is again due to the dominance of penalty terms from unmatched bounding boxes. For a comparison across different

Lastly, we present the average precision and recall results using the COCO evaluation API [Lin et al., 2014] in Table 7.2. The COCO evaluation API has been widely used in existing literate to evaluate the performance of object detectors. However, we must note that the average precision and recall results do not evaluate the probabilistic aspect of the classification performance, unlike the NLL and ECE metrics shown earlier in this section. From the results in Table 7.2, we observe that the full ensemble with average merging performs the best in terms of average precision and recall, however, the gains are still somewhat modest as compared to the respective base models. When comparing across models and methods, we find that AdaMixer full ensemble with average merging technique performs the best.

7.4.4 Experiment 3: Evaluating bounding box coverage and precision

In this experiment, we focus on evaluating the coverage and precision metrics for different methods and merging strategies. Similarly to the previous experiment, we begin by evaluating the effect of the merging strategy. We present the results analyzing coverage and precision for different merging strategies in Figures 7.14 - 7.17.

Table 7.2. Comparison of average precision and recall results using COCO evaluation API.

	Average Precision	Average Recall
Base DETR	0.386	0.480
Base AdaMixer	0.400	0.503
DETR Decoder Ensemble - Average	0.375	0.464
DETR Decoder Ensemble - Maximal	0.355	0.440
DETR Decoder Ensemble - Minimal	0.336	0.416
AdaMixer Decoder Ensemble - Average	0.399	0.503
AdaMixer Decoder Ensemble - Maximal	0.375	0.475
AdaMixer Decoder Ensemble - Minimal	0.367	0.465
DETR Full Ensemble - Average	0.399	0.494
DETR Full Ensemble - Maximal	0.349	0.439
DETR Full Ensemble - Minimal	0.332	0.419
AdaMixer Full Ensemble - Average	0.408	0.518
AdaMixer Full Ensemble - Maximal	0.373	0.473
AdaMixer Full Ensemble - Minimal	0.364	0.463

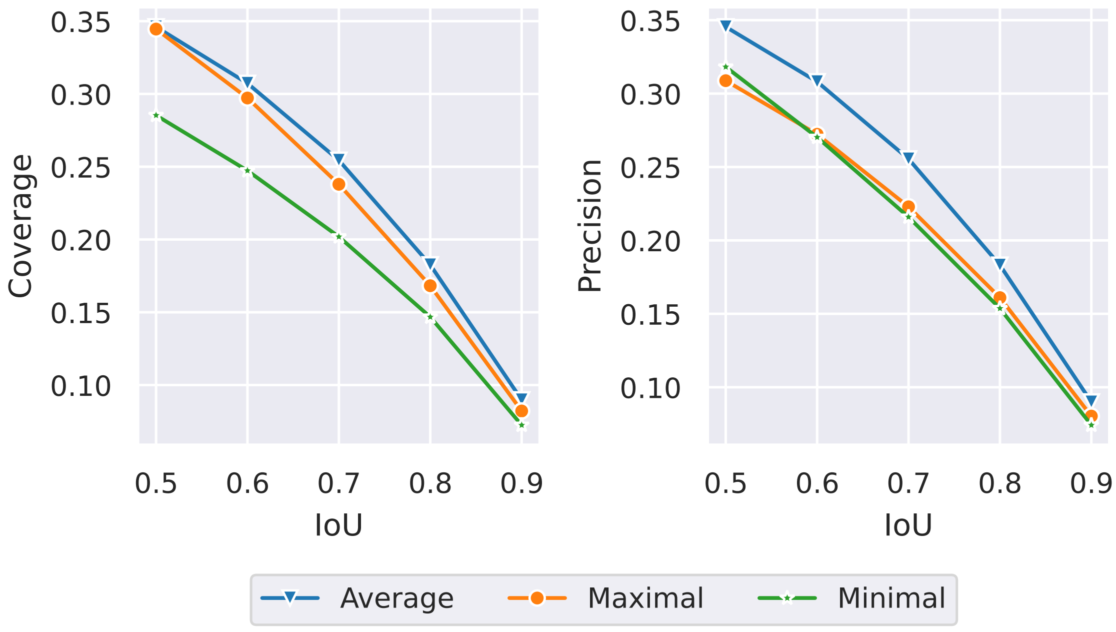
Figures 7.14,7.16 show the performance comparison for the entire validation data set, including penalties for unmatched bounding boxes, while Figures 7.15,7.17 show the performance metrics for the matched bounding boxes. Similar to the classification metrics, we find that while evaluating the performance on the entire validation dataset, the average merging strategy performs better than the other two merging strategies. As we observed in the previous experiment, this is potentially due to the larger number of unmatched ground truth bounding boxes, which lead to a higher penalty on the performance metrics.

Among the results presented for only matched bounding boxes in Figures 7.15 & 7.17 we observe that the maximal merging strategy achieves higher coverage than the rest, while the minimal merging strategy achieves higher precision than the rest. This is anticipated, as the maximal merging strategy inherently generates larger bounding boxes, and is thus bound to have a higher coverage. On the other hand, the minimal merging strategy is more conservative, producing smaller bounding boxes and therefore leads to higher precision. It is also important to note that the maximal and minimal merging strategy also leads to the lowest performance in precision and coverage metrics, respectively. The average merging strategy has a performance level between the maximal and the minimal merging strategy, as it produces bounding boxes that are smaller than the maximal strategy, but not as conservative as the minimal merging strategy.

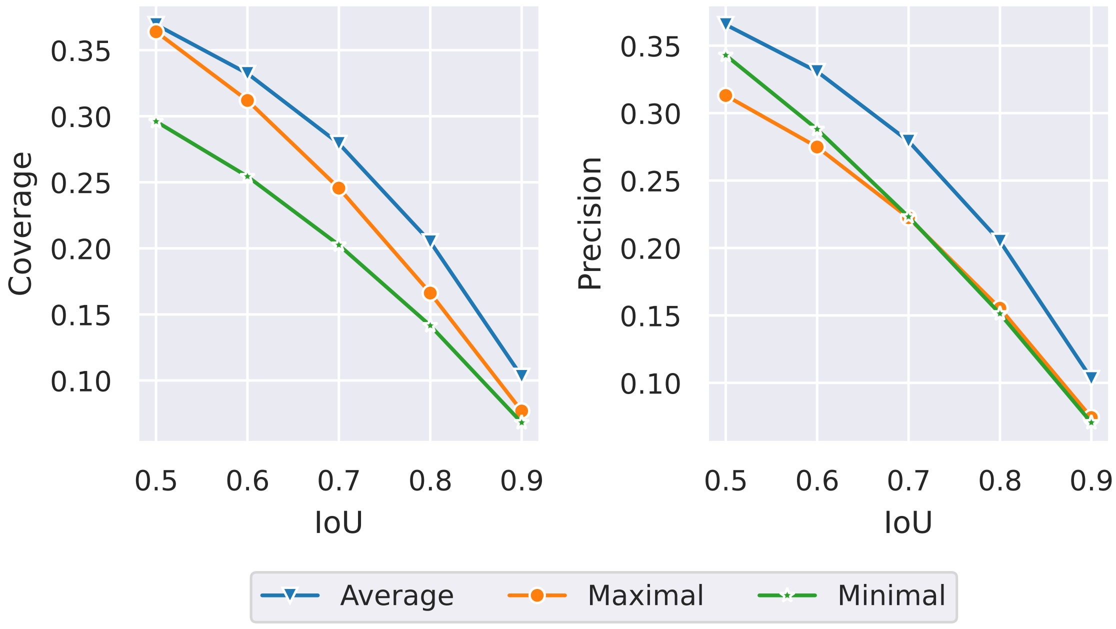
Finally, we present the results that compare the precision and coverage metrics between different approximate inference methods in Figures 7.18 & 7.19. From the results in Figure 7.18, we note that the deep ensembles achieve better coverage and precision over the base models. This is a promising result, as it shows that the average bounding box generated after merging does a better job of capturing the ground-truth bounding boxes than an individual model. As we increase the matching IoU threshold, we observe that the performance gap closes between the base model and the ensemble. Similar to the previous experiment, this is largely due to the fact that the number of matched bounding boxes decreases and the penalty terms generated from unmatched bounding boxes begin to dominate the metric computation. Finally, we also note that all the methods perform very similar when it comes to the precision metric on matched bounding boxes.

7.4.5 Experiment 4: Evaluating Objectness Uncertainty

In this experiment, we evaluate the objectness uncertainty across different approximate inference methods and merging strategies. As we describe in Section 7.3.4, our goal is to evaluate how the models perform on the binary classification task of identifying whether a local region belongs to any object or not. We introduce additional methods for this task that look at composing the objectness information from deep ensembles without clustering. To do this, we simply generate an objectness map for each member of the ensemble, and while composing the maps, for every pixel, we average

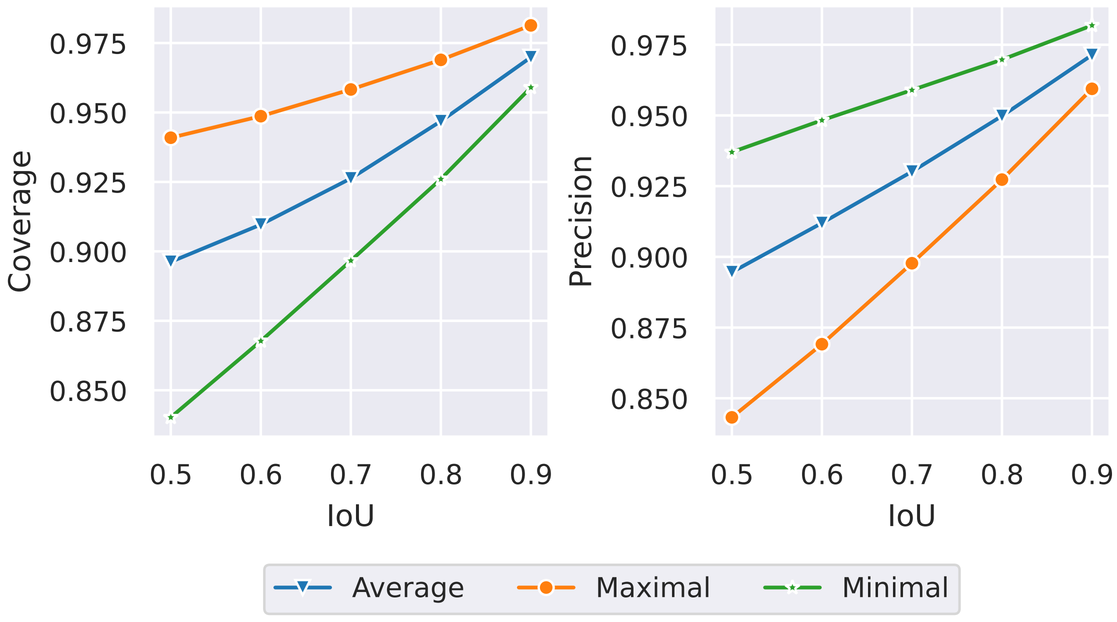


(a) Decoder Ensemble

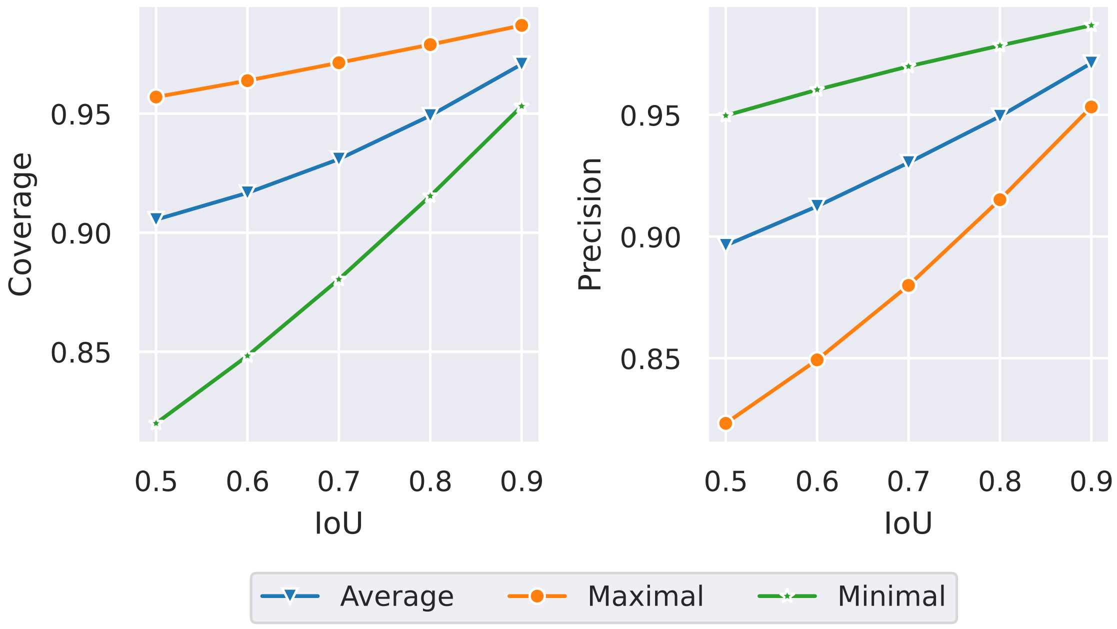


(b) Full Ensemble

Figure 7.14. Performance comparison of different merging strategies for decoder ensemble and full ensemble on the entire validation set for DETR model. We present the bounding box coverage (↑), and precision (↑) for different matching IoU thresholds.

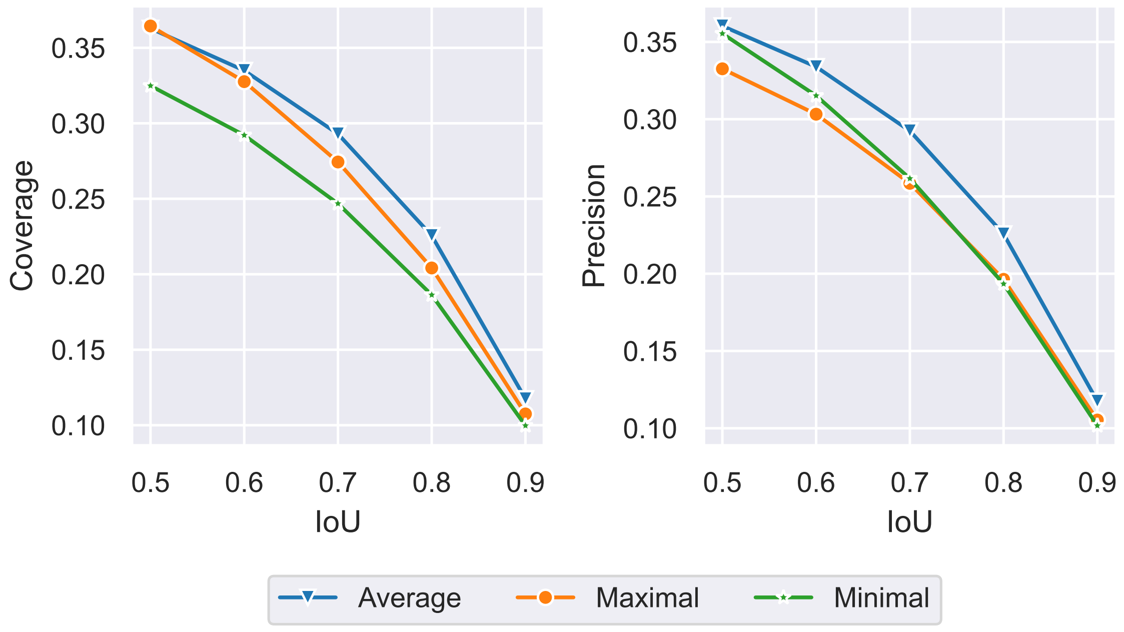


(a) Decoder Ensemble

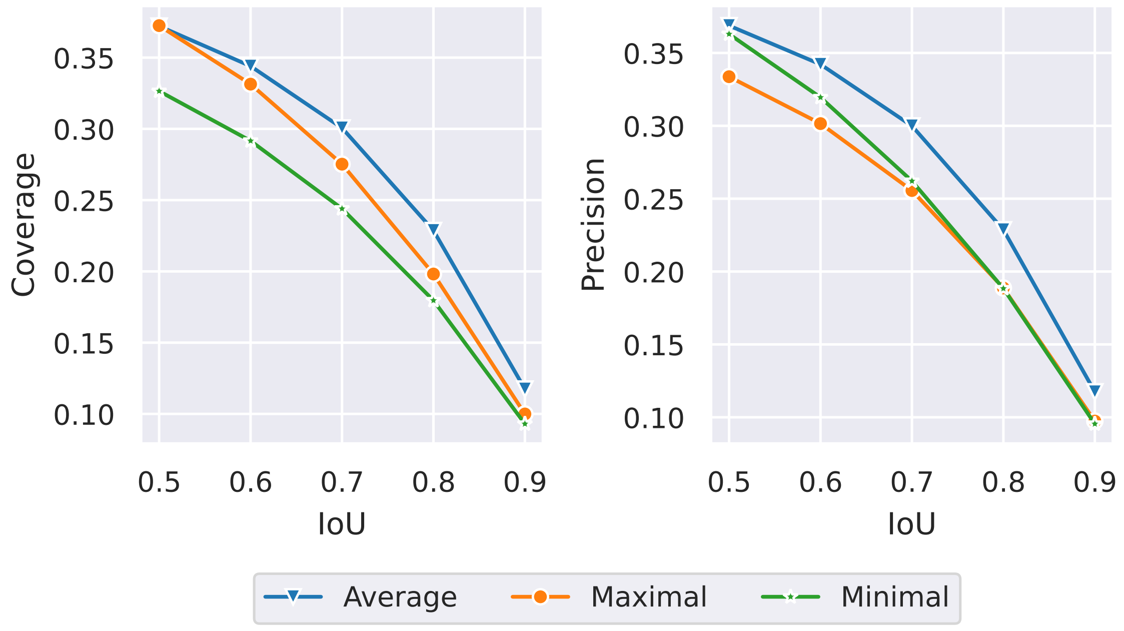


(b) Full Ensemble

Figure 7.15. Performance comparison of different merging strategies for decoder ensemble and full ensemble on matched bounding boxes only for DETR model. We present the bounding box coverage (↑), and precision (↑) for different matching IoU thresholds.

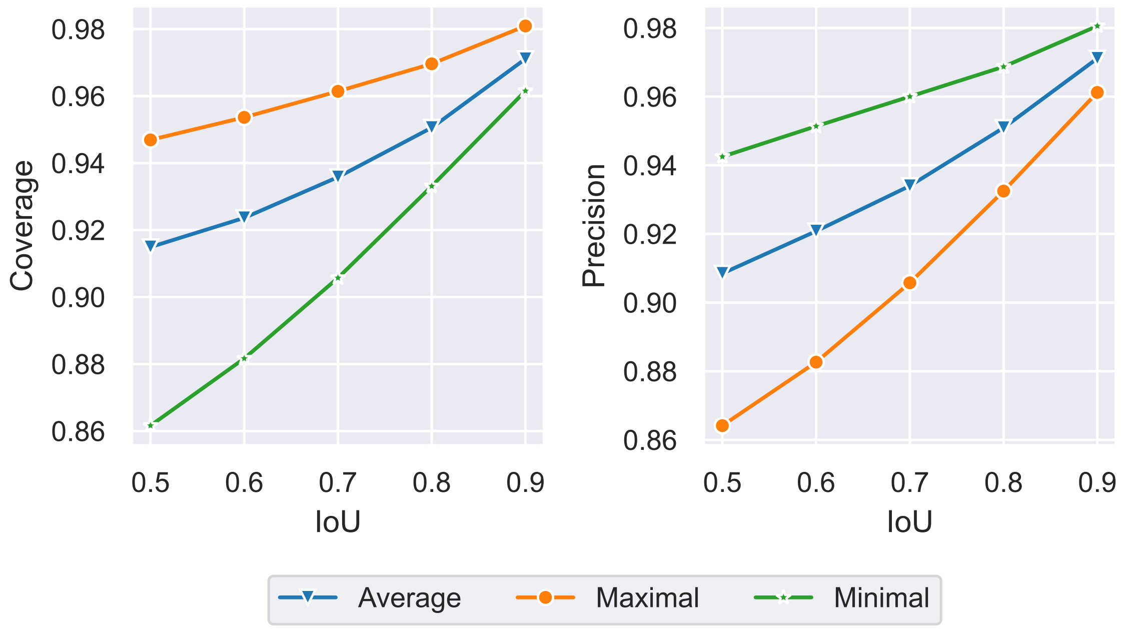


(a) Decoder Ensemble

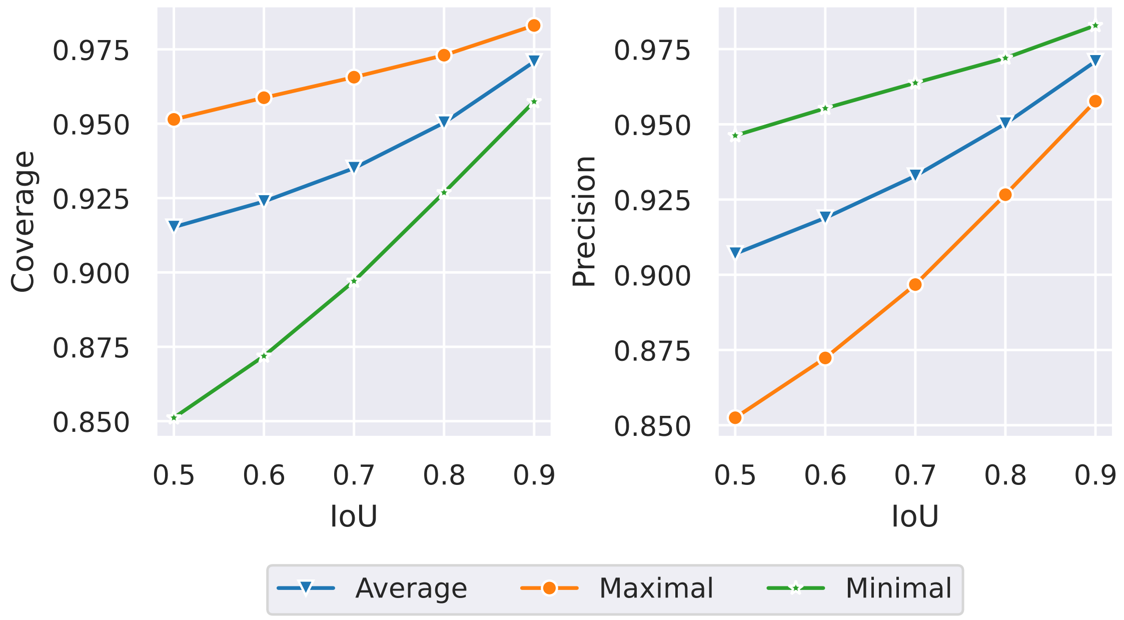


(b) Full Ensemble

Figure 7.16. Performance comparison of different merging strategies for decoder ensemble and full ensemble on the entire validation set for AdaMixer model. We present the bounding box coverage (↑), and precision (↑) for different matching IoU thresholds.

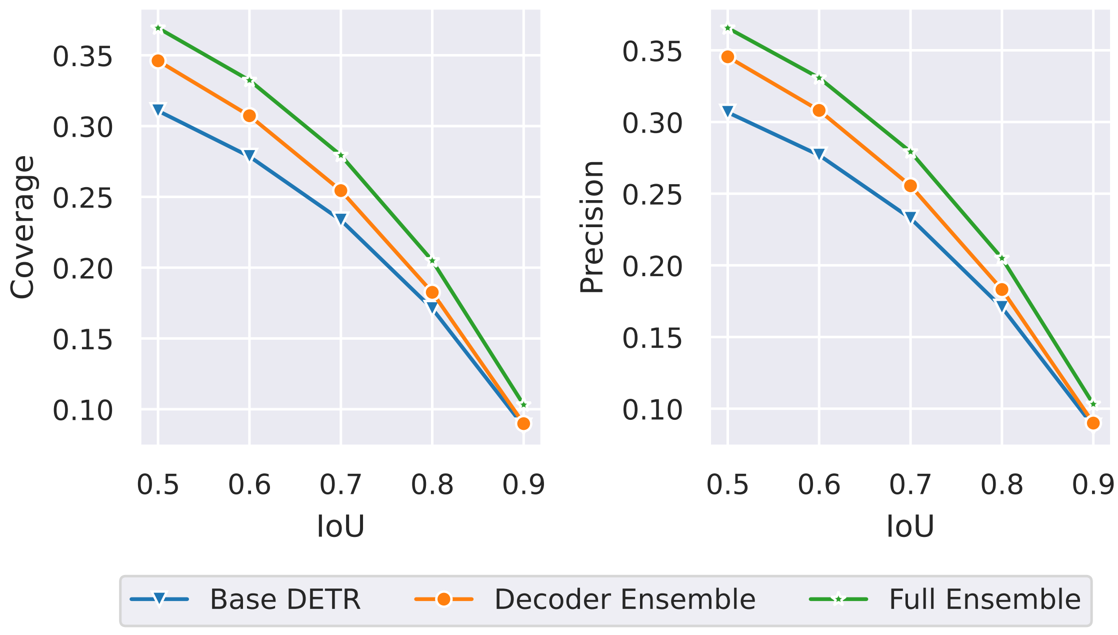


(a) Decoder Ensemble

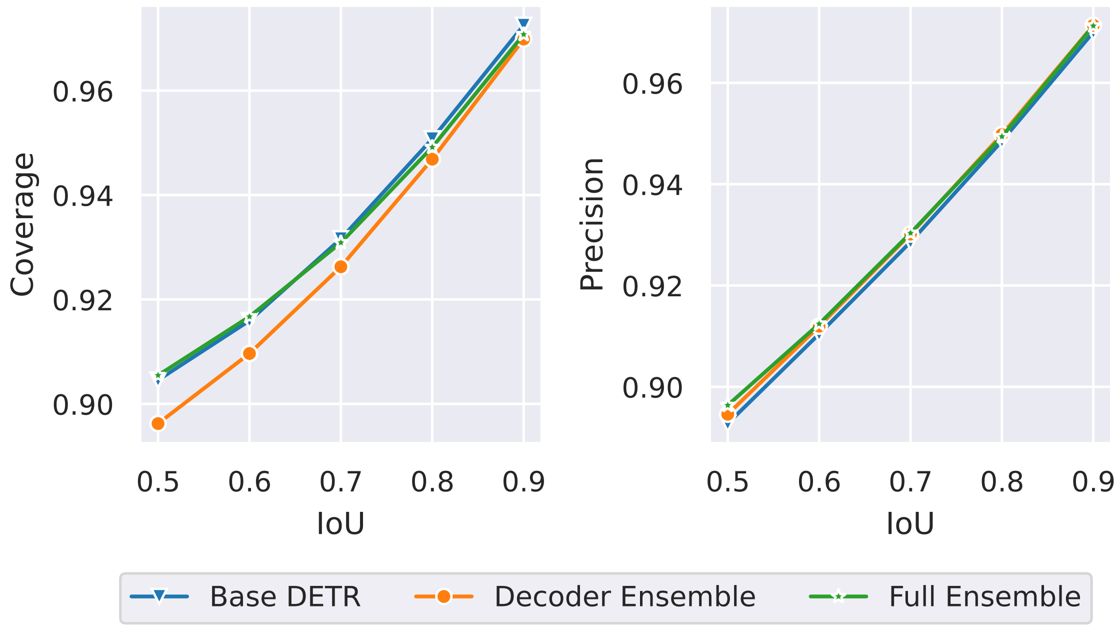


(b) Full Ensemble

Figure 7.17. Performance comparison of different merging strategies for decoder ensemble and full ensemble on matched bounding boxes only for AdaMixer model. We present the bounding box coverage (↑), and precision (↑) for different matching IoU thresholds.

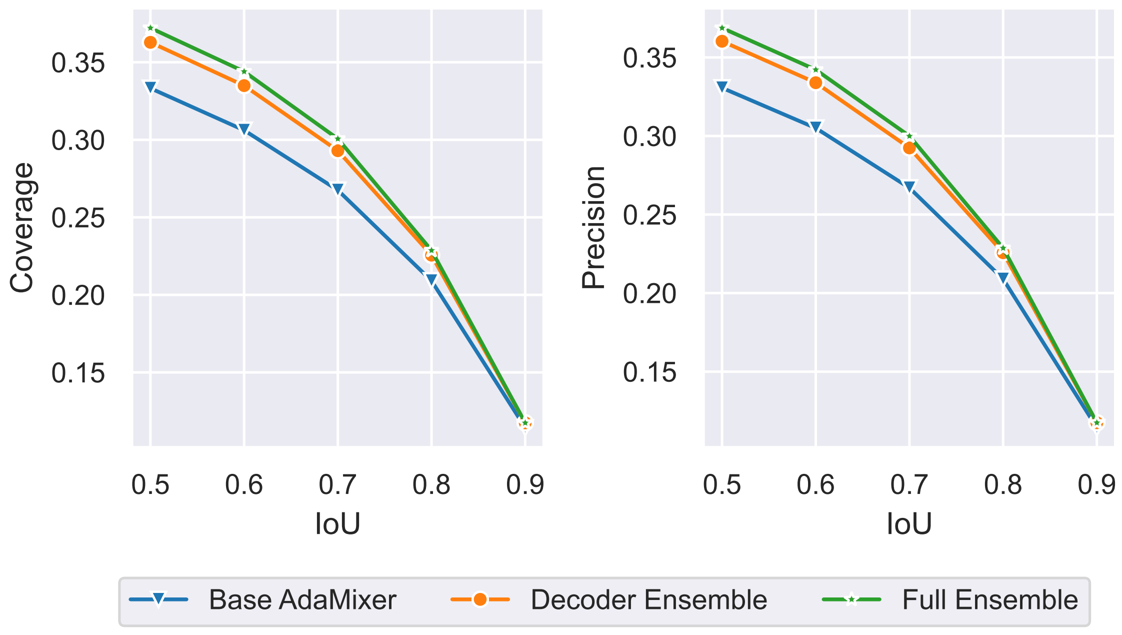


(a) All data.

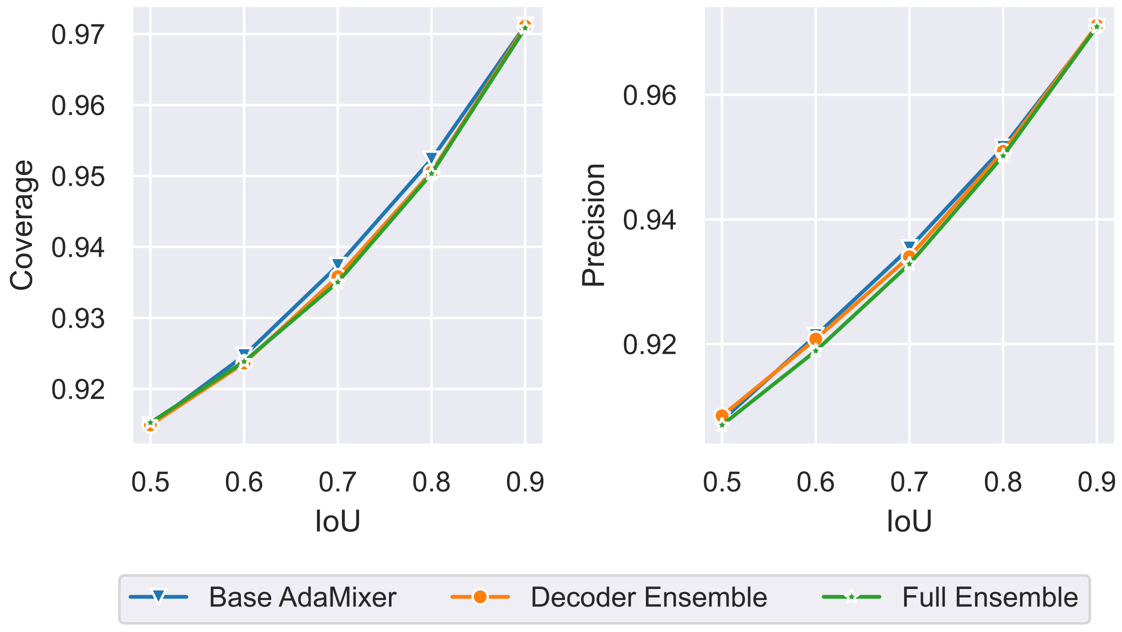


(b) Matched boxes only.

Figure 7.18. A comparison of bounding box coverage and precision across different methods for DETR model. We present coverage (↑), and precision (↑) for different matching IoU thresholds. For deep ensembles, we use the average bounding box merging strategy.



(a) All data.



(b) Matched boxes only.

Figure 7.19. A comparison of bounding box coverage and precision across different methods for AdaMixer model. We present coverage (↑), and precision (↑) for different matching IoU thresholds. For deep ensembles, we use the average bounding box merging strategy.

the objectness score across all the members of the ensemble. The results of this experiment are presented in Table 7.3. Note that for the final downsampling step of the image grids using max-pooling, we use a kernel of size (4, 4).

Table 7.3. Objectness Uncertainty Results.

Method	Merging Strategy	NLL	Accuracy	AUROC	Precision	Recall	F1
Base DETR	-	0.313	0.916	0.959	0.865	0.958	0.909
Base AdaMixer	-	0.369	0.913	0.955	0.864	0.952	0.906
DETR Decoder Ensemble	Average	0.317	0.914	0.954	0.866	0.953	0.907
	Maximal	0.304	0.909	0.956	0.848	0.969	0.904
	Minimal	0.383	0.910	0.942	0.878	0.923	0.900
AdaMixer Decoder Ensemble	Average	0.319	0.911	0.951	0.861	0.951	0.904
	Maximal	0.298	0.904	0.954	0.839	0.969	0.899
	Minimal	0.411	0.904	0.935	0.875	0.912	0.893
DETR Full Ensemble	Average	0.289	0.917	0.958	0.865	0.961	0.911
	Maximal	0.287	0.907	0.959	0.837	0.980	0.903
	Minimal	0.390	0.909	0.940	0.881	0.917	0.899
AdaMixer Full Ensemble	Average	0.305	0.913	0.954	0.863	0.954	0.906
	Maximal	0.286	0.904	0.956	0.834	0.976	0.899
	Minimal	0.418	0.904	0.934	0.878	0.907	0.893
DETR Decoder Ensemble	No merging	0.261	0.921	0.968	0.878	0.953	0.914
DETR Full Ensemble	No merging	0.205	0.928	0.977	0.888	0.956	0.921
AdaMixer Decoder Ensemble	No merging	0.238	0.921	0.969	0.884	0.945	0.913
AdaMixer Full Ensemble	No merging	0.214	0.926	0.974	0.884	0.951	0.919

There are a few key takeaways from this experiment. First, when evaluating the probabilistic aspect of the methods using NLL and AUROC, we observe that the full ensemble without merging predicted bounding boxes performs much better than the rest of the methods. Specifically, we see that both decoder ensemble and full ensemble without merging perform better than their counterparts which include merging. This is intuitive, as the merging step can lead to filtering information if a cluster does not satisfy the minimum number of detections threshold. Averaging the most confident objectness probability scores across multiple members of the ensemble can lead to small objectness probability mass assigned to areas of the image where an individual model does not assign. Furthermore, the reverse is also true. In cases where an individual model has an overconfident prediction, averaging the objectness scores can also help reduce the prediction probability to a more moderate level. An example demonstrating this is shown in Figure 7.20.

Next, when comparing between merging strategies, we find that the minimal merging strategy achieves the lowest performance among other merging techniques. Again, this is intuitive, as the minimal bounding-box-generating strategy would assign objectness probability to fewer pixels in the image. Comparing the full ensemble and the decoder ensemble, except for the minimal merging strategy, the full ensemble tends to perform better for a given merging strategy.

7.4.6 Experiment 5: Runtime-aware Performance Comparison

For practical applications, it is vital to understand the prediction latency. In this experiment, we evaluate the prediction latency of the methods introduced in this chapter on the classification task. In the previous experiments, we used a consistent ensemble size of 9 models for both the decoder ensemble and the full ensemble. However, for the decoder ensemble, since the backbone is fixed, we can cache the outputs from the encoder/backbone, and run them through all 9 decoders. This helps us to reduce the overall prediction latency. We give a detailed breakdown of the latency numbers in Table 7.4. We note that the results presented in Table 7.4 are approximate runtime numbers and that they do not account for additional overheads such as the time required to load models in the memory. Furthermore, the runtime numbers for ensembles are obtained using linear extrapolation over the results presented for a

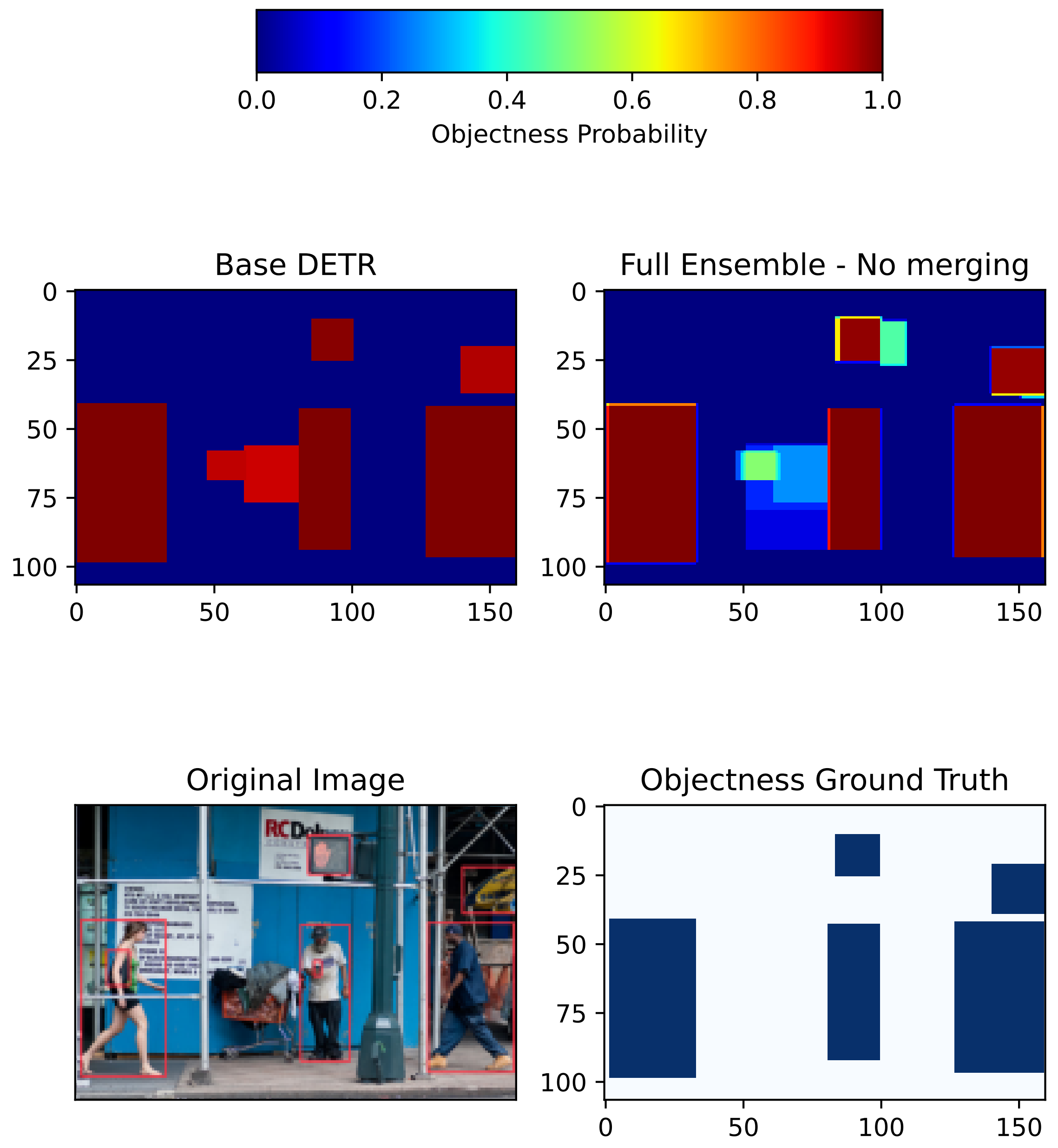


Figure 7.20. (Top, Left) Objectness score map using full ensemble (without merging). (Top, Right) Objectness score map using Base DETR. (Bottom, Left) Original image with ground truth bounding boxes. (Bottom, Right) Ground truth objectness map. The blue area corresponds to objects, and the white area corresponds to the background. Note that the base DETR model assigns a strong objectness probability to the shopping cart in the middle of the image, which does not have a ground truth bounding box assigned to it. However, the full ensemble helps moderate the confidence due to the averaging of the scores.

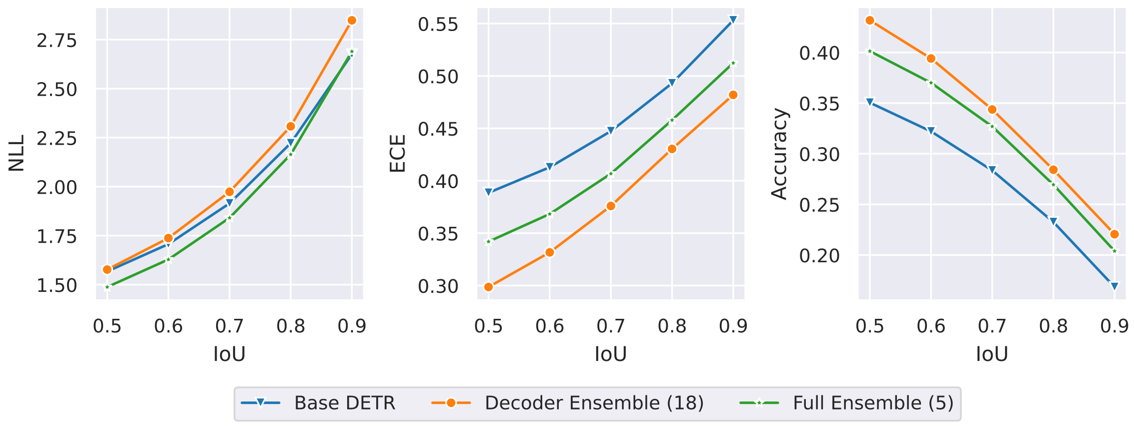
single model. We also provide frames/sec (FPS) results based on the total runtime latency.

Table 7.4. Runtime latency results. Median runtime latency numbers are presented as average milliseconds per image on the validation dataset. The number of models in the ensemble is indicated in parentheses. We use an NVIDIA 3080Ti GPU to evaluate the runtime latency for all models.

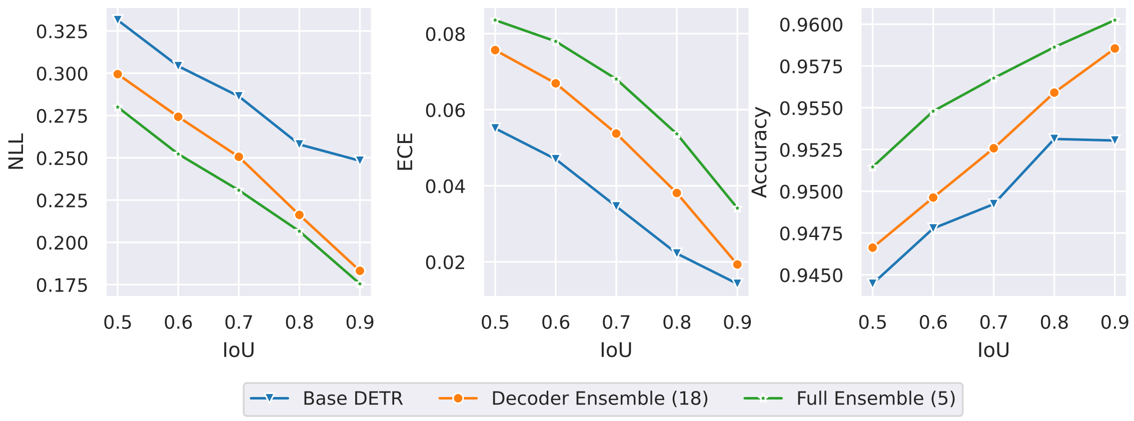
	Backbone	Encoder	Decoder	Clustering	Total	FPS
Base DETR	15.5	4.28	5.25	-	25.0	40.0
Base AdaMixer	17.7	-	16.4	-	34.1	29.7
DETR Decoder Ensemble (9)	15.5	4.28	47.3	7.1	74.1	13.5
DETR Decoder Ensemble (18)	15.5	4.3	94.5	28.3	142.6	7.0
DETR Full Ensemble (5)	77.4	21.4	26.3	2.9	127.9	7.8
DETR Full Ensemble (9)	139.3	38.5	47.3	6.3	231.4	4.3
AdaMixer Decoder Ensemble (9)	17.7	-	147.6	7.2	172.5	5.8
AdaMixer Full Ensemble (9)	159.3	-	147.6	7.2	314.1	3.2

For this experimental section, we compare results from a full ensemble consisting of 5 DETR models and a decoder ensemble that consists 18 DETR models. As noted in Table 7.4, the runtime of these are very similar, and thus they make for a more fair runtime-aware comparison. For this experiment, we focus on the classification performance and the coverage and precision of the predicted bounding boxes. Additionally, we focus specifically on the average bounding box merging strategy. For merging, we use the same IoU threshold of 0.75 for grouping and ensure the same fraction of the total ensemble as the threshold of the minimum number of detections by setting the minimum number of detections for a cluster to 2 for the 5-member full ensemble and 7 for the 18-member decoder ensemble [Miller et al., 2021]. The results of the classification performance are presented in Figure 7.21, and the results for the coverage and precision of the predicted bounding boxes are presented in Figure 7.22.

We observe that the DETR full ensemble still performs better than the DETR decoder ensemble on the NLL metric. Looking at ECE and accuracy on the entire validation dataset, we observe that the DETR decoder ensemble has better performance, while

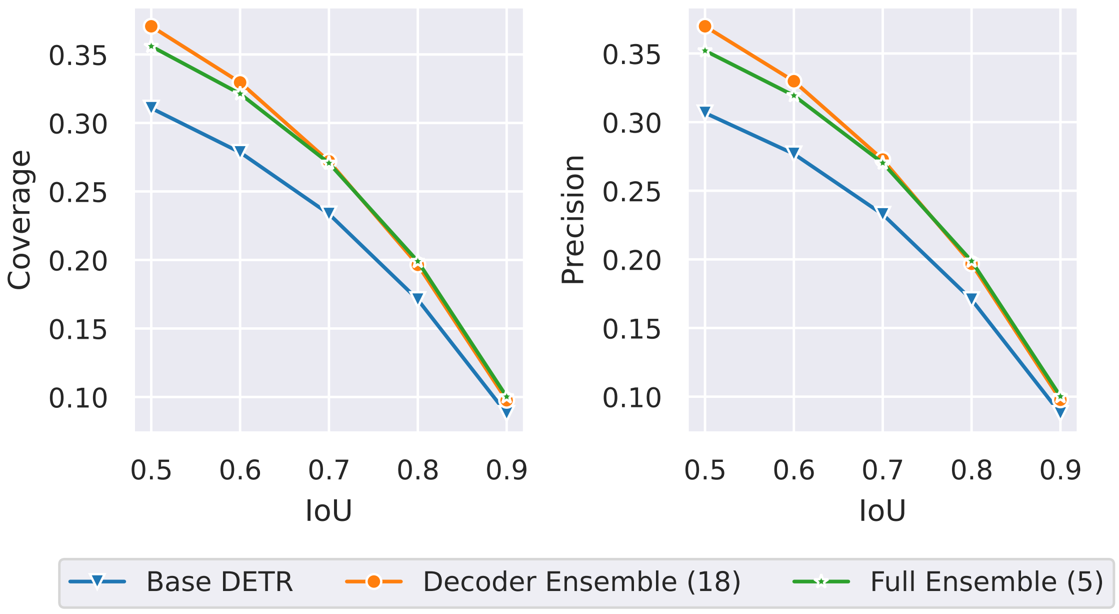


(a) All data

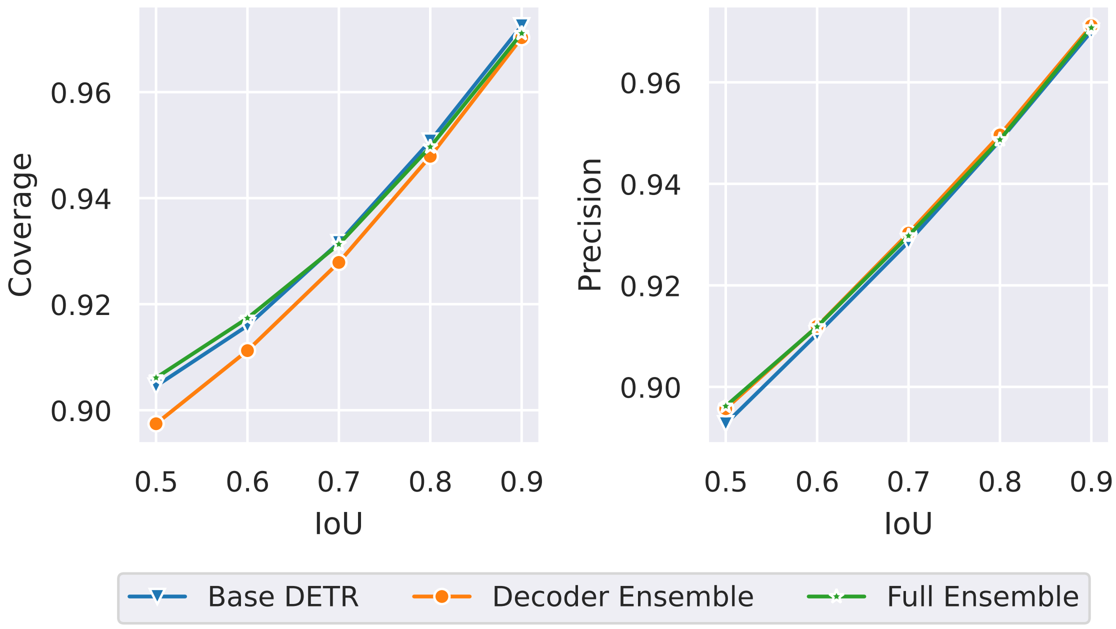


(b) Matched boxes only

Figure 7.21. A comparison of classification performance metrics across different methods. We present negative log-likelihood (NLL, ↓), expected calibration error (ECE, ↓), and accuracy (↑) for different matching IoU thresholds. For deep ensembles, we use the average bounding box merging strategy.



(a) All data.



(b) Matched boxes only.

Figure 7.22. A comparison of bounding box coverage and precision across different methods. We present coverage (↑), and precision (↑) for different matching IoU thresholds. For deep ensembles, we use the average bounding box merging strategy.

on the matched bounding boxes, the full ensemble tends to yield better accuracy and ECE when compared with the decoder ensemble. This is especially an interesting observation, as the accuracy and ECE improvements on the larger decoder ensemble on the full validation set do not necessarily translate to an improvement in NLL.

On the coverage and precision of the predicted bounding boxes, we observe that the decoder ensemble performs similarly (in fact slightly better on lower IoU thresholds) to the full ensemble when we look at all the validation data. Similarly, on the matched bounding boxes, we observe that all the different methods perform similarly.

7.5 Conclusions

In this chapter, we have explored the large-scale application of uncertainty quantification methods to object detection using object-query based object detectors. We presented a range of evaluation principles of interest and an empirical analysis evaluating different combinations of the methods presented on said evaluation principles. Through this empirical analysis, we show how parameter uncertainty estimation using deep ensembles can help us improve on class uncertainty, location uncertainty, and objectness uncertainty. Furthermore, we also present runtime latency results and show how a larger decoder ensemble with similar latency to a smaller full ensemble can lead to interesting tradeoffs among different performance metrics.

Several important directions for future work emerge from this chapter. First, while looking at location uncertainty we have focused on uncertainty for the center location, however, this can be expanded to include uncertainty around the height and width of the object as discussed in the related work section. Second, deep ensembles based object detectors are expensive from computation and storage perspective. An interesting technique that allows us to explore the variability of model predictions is using test time augmentation for images [Ayhan and Berens, 2018]. Using test-time augmentation, we can generate an ensemble of images, and get model predictions using a single model. Note that this still requires us to perform multiple forward passes through the model, but we only have to store one copy of model parameters. Additional future directions include expanding the set of approximate inference techniques, exploring other merging techniques and the underlying affinity scores used in merging, as well as looking at other similar model architectures.

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

8.1 Conclusions

The central theme of the problems explored in this thesis is how different approximate Bayesian deep learning methods trade off performance against computational scalability. We first provided a review of some prominent methods in Bayesian deep learning while commenting on the scalability challenges they face. To tackle the challenge of computational scalability, we also provided a primer on existing methods of model pruning and knowledge distillation, which can be coupled with existing methods in Bayesian deep learning.

Next, we introduced URSABench, a Bayesian deep learning benchmarking framework operationalizing a set of important evaluation principles for evaluating the probabilistic nature of models, such as negative log likelihood, calibration performance, OOD detection performance, decision-making performance, and misclassification detection performance in addition to accuracy. URSABench has been built with the goal of extensibility and ease of use in mind so that practitioners working with point-estimated neural network models can smoothly extend them to Bayesian neural networks. We provide additional tools for practitioners to deploy models on edge devices and evaluate their latency performance. We also provided a set of initial benchmark results that evaluate a set of existing approximate inference techniques across a range of models and datasets.

We then turned our focus towards compressing the parameter posterior using knowledge distillation. We introduced a general framework for distilling the posterior expectations of a teacher model to a student model. Through an extensive empirical evaluation, we demonstrated that distillation is highly dependent on the level of posterior uncertainty, as well as student model architecture. We also showed that our framework performs better overall than the ensemble distribution distillation baseline [Malinin et al., 2020] on two uncertainty quantification applications: out-of-distribution detection, and uncertainty-based ranking.

On the topic of posterior compression, we also explored sparse model substructures in approximate Bayesian inference. We leveraged existing approaches for generating sparse substructures using iterative pruning and evaluated the performance of stochastic gradient MCMC methods on the sparse substructures. We also looked at randomly generated substructures as an alternate to the computationally expensive procedure of iterative pruning and observed that while substructures generated using iterative pruning tend to give better performance at higher sparsity rates, the randomly generated spare substructures can perform as well on a spectrum of sparsity rates.

We net explored additional applications aside from classification. Among the additional applications, we first looked at the application of decision-making using Bayesian decision theory. Under Bayesian decision theory, we utilize the posterior predictive distribution to compute the expected utility/cost of all the decisions possible and pick a decision that maximizes the utility (or minimizes the decision cost). To this end, we introduced a post-hoc posterior predictive correction framework, with the goal of encouraging the corrected predictive distribution to produce high utility decisions. Our framework is agnostic to the choice of underlying approximate inference technique and produces decisions in constant time. Through extensive experimental evaluations, we showed that our approach leads to better decision-making on a range of downstream tasks compared to baseline methods.

Finally, we also looked at the application of approximate Bayesian inference to the task of computer vision-based object detection. Object detectors have a more complex output space than the models used in earlier chapters of this thesis, as they produce multiple bounding boxes, each containing the location information and class distribution. As a result, the parameter uncertainty can be transformed into different forms of output uncertainty, such as classification uncertainty, location uncertainty, and objectness uncertainty. We introduced a set of evaluation principles for the task looking at these aspects of uncertainty and demonstrated the utility of deep ensembles-based uncertainty estimation on the detection transformer model. We also provided estimated runtime latency results with a detailed breakdown across different model and method components, along with a latency-aware comparison between full ensembles and decoder ensembles.

8.2 Future Directions

There are several future directions motivated by the contributions of this thesis. At the core of all methods and applications presented in this thesis lie approximate Bayesian inference techniques. Better approximate inference techniques can lead us to a better exploration of the parameter posterior, which can further lead us to improve downstream application performance. A critical challenge in Bayesian deep learning is the high dimensionality of the parameter space. As a part of URSABench, we evaluated subspace inference using PCA-based subspace and ESS as the sampling algorithm. There has been recent work that examines approximate variational inference in a low dimensional space, using rank-1 factors [Dusenberry et al., 2020]. This can be easily extended to stochastic gradient MCMC algorithms and is a promising direction to pursue. A limitation of this thesis is that we solely focus on isotropic Gaussian priors. Indeed, there has been some work that looks at the impact of various priors for Bayesian neural networks [Novak et al., 2019, Fortuin et al., 2021, Izmailov et al., 2021a]. A useful future direction would be to examine these different priors in the context of the applications presented in this thesis, such as decision-making, and object detection. To make URSABench more exhaustive in the future, a useful future thread to pursue would be to incorporate all the aforementioned methods within URSABench.

As noted earlier in this thesis, an important use case of Bayesian neural networks is the decomposition of total uncertainty into epistemic and aleatoric components. The experiments evaluating the epistemic uncertainty for applications such as out-of-distribution detection and misclassification detection are based on the regimes where we have thousands of labelled instances during training. Thus, it is important to investigate the quality of the epistemic and total uncertainty estimates in the regime of a low number of labelled training instances.

In this thesis, except for the task of object detection, the bulk of the focus has been on modeling categorical output spaces, either in decision-making or classification. However, it is equally common to encounter applications that require modeling continuous outputs (i.e. $\mathcal{Y} \subset \mathbb{R}^D \text{or} \mathcal{A} \subset \mathbb{R}^D, D \in \mathbb{N}$). A useful set of future directions would include extending the methods and empirical evaluations proposed in this thesis to the case of modeling continuous output variables. For example, a practical application that builds on object detection is estimating the number of objects present in a given scene. While this can be formulated as count regression, a more Bayesian treatment would involve building a mixture model, where each member of the mixture would be a sample from the parameter posterior.

Understanding the computational challenges faced in deploying Bayesian neural networks is also an important focus area of this thesis. While working with object detection models, we have not explored the idea of compressing the posterior representation, which can hinder their practical deployment due to the large model size. Given the unique nature of the outputs of object detection models such as DETR, it would be an interesting future direction to look at distilling the posterior representation from these models into a student model to improve prediction latency.

BIBLIOGRAPHY

Ehsan Abbasnejad, Justin Domke, and Scott Sanner. Loss-calibrated monte carlo action selection. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Bogdan Alexe, Thomas Deselaers, and Vittorio Ferrari. Measuring the objectness of image windows. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2189–2202, 2012.

Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2270–2278, 2016.

Murat Seckin Ayhan and Philipp Berens. Test-time data augmentation for estimation of heteroscedastic aleatoric uncertainty in deep neural networks. In *Medical Imaging with Deep Learning*, 2018. URL `https://openreview.net/forum?id=rJZz-knjz`.

Vijay Badrinarayanan, Alex Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:2481–2495, 2017.

Anoop Korattikara Balan, Vivek Rathod, Kevin P Murphy, and Max Welling. Bayesian dark knowledge. In *Advances in Neural Information Processing Systems*, pages 3438–3446, 2015.

Maximilian Balandat, Brian Karrer, Daniel R Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. Botorch: Programmable bayesian optimization in pytorch. *arXiv preprint arXiv:1910.06403*, 2019.

James O. Berger. *Statistical Decision Theory and Bayesian Analysis, Second Edition*. Springer, 1985. ISBN 3-540-96098-8. doi: 10.1007/978-1-4757-4286-2. URL `https://doi.org/10.1007/978-1-4757-4286-2`.

Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep universal probabilistic programming. *J. Mach. Learn. Res.*, 20:28:1–28:6, 2019. URL `http://jmlr.org/papers/v20/18-403.html`.

Pier Giovanni Bissiri, Chris C Holmes, and Stephen G Walker. A general framework for updating belief distributions. *Journal of the Royal Statistical Society. Series B, Statistical methodology*, 78(5):1103, 2016.

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2:129–146, 2020.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1613–1622, Lille, France, 07–09 Jul 2015. PMLR. URL `https://proceedings.mlr.press/v37/blundell15.html`.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

Glenn W Brier. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 78(1):1–3, 1950.

Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of markov chain monte carlo*. CRC press, 2011.

Gabriel J. Brostow, Julien Fauqueur, and Roberto Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, xx(x): xx–xx, 2008a.

Gabriel J. Brostow, Jamie Shotton, Julien Fauqueur, and Roberto Cipolla. Segmentation and recognition using structure from motion point clouds. In *ECCV*, pages 44–57, 2008b.

Yaroslav Bulatov. notmnist dataset. 2011.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pages 213–229. Springer, 2020.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. *2017 IEEE Symposium on Security and Privacy (SP)*, 2017.

Carlos M Carvalho, Nicholas G Polson, and James G Scott. Handling sparsity via the horseshoe. In *Artificial Intelligence and Statistics*, pages 73–80. PMLR, 2009.

George Casella and Edward I George. Explaining the gibbs sampler. *The American Statistician*, 46(3):167–174, 1992.

Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019.

Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic gradient hamiltonian monte carlo. In *International conference on machine learning*, pages 1683–1691. PMLR, 2014a.

Tianqi Chen, Emily B. Fox, and Carlos Guestrin. Stochastic gradient hamiltonian monte carlo. In *International Conference on Machine Learning*, 2014b.

Ming-Ming Cheng, Ziming Zhang, Wen-Yan Lin, and Philip Torr. Bing: Binarized normed gradients for objectness estimation at 300fps. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3286–3293, 2014.

Siddhartha Chib and Edward Greenberg. Understanding the metropolis-hastings algorithm. *The american statistician*, 49(4):327–335, 1995.

Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature. *arXiv:1812.01718*, 2018a.

Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature, 2018b.

Adam Coates, Andrew Y. Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS*, 2011.

Adam D. Cobb, Stephen J. Roberts, and Yarin Gal. Loss-calibrated approximate inference in bayesian neural networks. In *ICML theory of deep learning workshop*, 2018a.

Adam D Cobb, Stephen J Roberts, and Yarin Gal. Loss-calibrated approximate inference in Bayesian neural networks. *Theory of deep learning workshop, ICML*, 2018b.

Adam D Cobb, Atılım Güneş Baydin, Andrew Markham, and Stephen J Roberts. Introducing an Explicit Symplectic Integration Scheme for Riemannian Manifold Hamiltonian Monte Carlo. *arXiv preprint arXiv:1910.06243*, 2019.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Decomposition of uncertainty for active learning and reliable reinforcement learning in stochastic systems. *ArXiv*, abs/1710.07283, 2017.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2018.

Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics letters B*, 195(2):216–222, 1987.

Michael Dusenberry, Ghassen Jerfel, Yeming Wen, Yian Ma, Jasper Snoek, Katherine Heller, Balaji Lakshminarayanan, and Dustin Tran. Efficient and scalable bayesian neural nets with rank-1 factors. In *International conference on machine learning*, pages 2782–2792. PMLR, 2020.

Vincent Fortuin, Adrià Garriga-Alonso, Florian Wenzel, Gunnar Rätsch, Richard Turner, Mark van der Wilk, and Laurence Aitchison. Bayesian neural network priors revisited. *arXiv preprint arXiv:2102.06571*, 2021.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Training pruned neural networks. *ArXiv*, abs/1803.03635, 2018.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

Mark Galassi, Jim Davies, James Theiler, Brian Gough, Gerard Jungman, Patrick Alken, Michael Booth, Fabrice Rossi, and Rhys Ulerich. Gnu scientific library. *Reference Manual. Edition 1.4, for GSL Version 1.4*, 2003.

Ziteng Gao, Limin Wang, Bing Han, and Sheng Guo. Adamixer: A fast-converging query-based object detector. In *CVPR*, 2022.

Soumya Ghosh and Finale Doshi-Velez. Model selection in bayesian neural networks via horseshoe priors. *arXiv preprint arXiv:1705.10388*, 2017.

Soumya Ghosh, Francesco Maria Delle Fave, and Jonathan Yedidia. Assumed density filtering methods for learning bayesian neural networks. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

Mark Girolami and Ben Calderhead. Riemann manifold langevin and hamiltonian monte carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214, 2011.

Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

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*, pages 2672–2680, 2014.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330, 2017.

Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances in Neural information processing systems*, 2016.

David Hall, Feras Dayoub, John Skinner, Haoyang Zhang, Dimity Miller, Peter Corke, Gustavo Carneiro, Anelia Angelova, and Niko Sünderhauf. Probabilistic object detection: Definition and evaluation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1031–1040, 2020.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *Advances in Neural information processing systems*, 2015.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016a. URL `http://arxiv.org/abs/1510.00149`.

Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Shijian Tang, Erich Elsen, Bryan Catanzaro, John Tran, and William J. Dally. Dsd: Regularizing deep neural networks with dense-sparse-dense training flow. *ArXiv*, abs/1607.04381, 2016b.

Ali Harakeh, Michael Smart, and Steven L Waslander. Bayesod: A bayesian approach for uncertainty estimation in deep object detectors. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 87–93. IEEE, 2020.

Babak Hassibi, David G. Stork, and Gregory J. Wolff. Optimal brain surgeon and general network pruning. *IEEE International Conference on Neural Networks*, pages 293–299 vol.1, 1993.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016b.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1398–1406, 2017.

Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=Hkg4TI9xl`.

Christian Henning, Johannes von Oswald, João Sacramento, Simone Carlo Surace, Jean-Pascal Pfister, and Benjamin F Grewe. Approximating the predictive distribution via adversarially-trained hypernetworks. In *Bayesian Deep Learning Workshop, NeurIPS (Spotlight) 2018*, 2018.

John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley Longman Publishing Co., Inc., USA, 1991. ISBN 0201503956.

Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

Conrad Holtsclaw, Meet P. Vadera, and Benjamin M. Marlin. Towards joint segmentation and active learning for block-structured data streams. In *The ACM SIGKDD Conference on Knowledge Discovery and Data Mining Workshop on Data Collection, Curation, and Labeling for Mining and Learning*, 2019.

Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2016.

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, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=BJYwwY9ll`.

Pavel Izmailov, Wesley J. Maddox, Polina Kirichenko, Timur Garipov, Dmitry P. Vetrov, and Andrew Gordon Wilson. Subspace inference for bayesian deep learning. In *UAI*, 2019.

Pavel Izmailov, Patrick Nicholson, Sanae Lotfi, and Andrew G Wilson. Dangers of bayesian model averaging under covariate shift. *Advances in Neural Information Processing Systems*, 34:3309–3322, 2021a.

Pavel Izmailov, Sharad Vikram, Matthew D Hoffman, and Andrew Gordon Gordon Wilson. What are bayesian neural network posteriors really like? In *International conference on machine learning*, pages 4629–4640. PMLR, 2021b.

Tommi S Jaakkola and Michael I Jordan. Bayesian parameter estimation via variational methods. *Statistics and Computing*, 10(1):25–37, 2000.

Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20:422–446, 2002.

Xiaojie Jin, Xiao-Tong Yuan, Jiashi Feng, and Shuicheng Yan. Training skinny deep neural networks with iterative hard thresholding methods. *ArXiv*, abs/1607.05423, 2016.

Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2): 183–233, 1999.

Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5574–5584, 2017. URL `https://proceedings.neurips.cc/paper/2017/hash/2650d6089a6d640c5e85b2b88265dc2b-Abstract.html`.

Brian W Kernighan and Dennis M Ritchie. *The C programming language.* 2006.

Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583, 2015.

Andreas Kirsch, Joost van Amersfoort, and Yarin Gal. Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 7024–7035, 2019.

Jeremias Knoblauch, Jack Jewson, and Theodoros Damoulas. Generalized variational inference: Three arguments for deriving new posteriors, 2019.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

David Krueger, Chin-Wei Huang, Riashat Islam, Ryan Turner, Alexandre Lacoste, and Aaron C. Courville. Bayesian hypernetworks. *ArXiv*, abs/1710.04759, 2017.

Tomasz Kuśmierczyk, Joseph Sakaya, and Arto Klami. Variational bayesian decision-making for continuous utilities. In *Advances in Neural Information Processing Systems*, pages 6395–6405, 2019.

Tomasz Kuśmierczyk, Joseph Sakaya, and Arto Klami. Correcting predictions for approximate bayesian inference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4511–4518, 2020.

Simon Lacoste-Julien, Ferenc Huszár, and Zoubin Ghahramani. Approximate inference for the loss-calibrated bayesian. In *AISTATS*, pages 416–424, 2011.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6402–6413, 2017.

Michael Truong Le, Frederik Diehl, Thomas Brunner, and Alois Knol. Uncertainty estimation for deep neural object detectors in safety-critical applications. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3873–3878. IEEE, 2018.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural information processing systems*, 1989.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *ArXiv*, abs/1608.08710, 2016.

Sheng Li, Jongsoo Park, and Ping Tak Peter Tang. Enabling sparse winograd convolution by native pruning, 2017.

Yingzhen Li and Richard E Turner. Rényi divergence variational inference. In *Advances in Neural Information Processing Systems 29*, pages 1073–1081. 2016.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

Christos Louizos and Max Welling. Multiplicative normalizing flows for variational bayesian neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2218–2227. JMLR. org, 2017.

Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. *ArXiv*, abs/1705.08665, 2017.

David JC MacKay. *Information theory, inference and learning algorithms.* Cambridge university press, 2003.

Wesley Maddox, Timur Garipov, Pavel Izmailov, Dmitry P. Vetrov, and Andrew Gordon Wilson. A simple baseline for bayesian uncertainty in deep learning. In *Advances in Neural information processing systems*, 2019.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018.

Andrey Malinin, Bruno Mlodozeniec, and Mark Gales. Ensemble distribution distillation. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=BygSP6Vtvr`.

Dimity Miller, Lachlan Nicholson, Feras Dayoub, and Niko Sünderhauf. Dropout sampling for robust object detection in open-set conditions. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3243–3249. IEEE, 2018.

Dimity Miller, Feras Dayoub, Michael Milford, and Niko Sünderhauf. Evaluating merging strategies for sampling-based uncertainty techniques in object detection. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2348–2354. IEEE, 2019.

Dimity Miller, Niko Sünderhauf, Michael Milford, and Feras Dayoub. Uncertainty for identifying open-set errors in visual object detection. *IEEE Robotics and Automation Letters*, pages 1–1, 2021. doi: 10.1109/LRA.2021.3123374.

Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey. *CoRR*, abs/2001.05566, 2020. URL `https://arxiv.org/abs/2001.05566`.

Thomas P Minka. Expectation propagation for approximate bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 362–369. Morgan Kaufmann Publishers Inc., 2001.

Kevin P Murphy. *Machine learning: a probabilistic perspective.* MIT press, 2012.

Iain Murray, Ryan P. Adams, and David J. C. MacKay. Elliptical slice sampling. In *AISTATS*, 2010.

Radford M. Neal. *Bayesian Learning for Neural Networks.* Springer-Verlag, Berlin, Heidelberg, 1996. ISBN 0387947248.

Radford M Neal. Slice sampling. *Annals of statistics*, pages 705–741, 2003.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011a.

Yuval Netzer, Tiejie Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. 2011b.

Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *International Conference on Machine Learning*, pages 625–632, 2005.

Roman Novak, Lechao Xiao, Yasaman Bahri, Jaehoon Lee, Greg Yang, Jiri Hron, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Bayesian deep convolutional networks with many channels are gaussian processes. In *ICLR*, 2019.

NVIDIA Corporation. TensorRT Developer Guide. URL `https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html`.

Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D Sculley, Joshua Nowozin, Sebastian Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift. In *Advances in Neural information processing systems*, pages 13969–13980, 2019.

Jongsoo Park, Sheng R. Li, Wei Wen, Ping Tak Peter Tang, Hai Li, Yiran Chen, and Pradeep Dubey. Faster cnns with direct sparse convolutions and guided pruning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=rJPcZ3txx`.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward 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 H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1137–1149, 2016.

Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable laplace approximation for neural networks. In *6th International Conference on Learning Representations, ICLR 2018-Conference Track Proceedings*, volume 6. International Conference on Representation Learning, 2018.

Christian Robert and George Casella. *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Adrian FM Smith and Gareth O Roberts. Bayesian computation via the gibbs sampler and related markov chain monte carlo methods. *Journal of the Royal Statistical Society: Series B (Methodological)*, 55(1):3–23, 1993.

Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 464–472. IEEE, 2017.

Hwanjun Song, Deqing Sun, Sanghyuk Chun, Varun Jampani, Dongyoon Han, Byeongho Heo, Wonjae Kim, and Ming-Hsuan Yang. Vidt: An efficient and effective fully transformer-based object detector. *CoRR*, abs/2110.03921, 2021. URL `https://arxiv.org/abs/2110.03921`.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Samuel Stanton, Pavel Izmailov, Polina Kirichenko, Alexander A Alemi, and Andrew G Wilson. Does knowledge distillation really work? *Advances in Neural Information Processing Systems*, 34:6906–6919, 2021.

Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23 (5):828–841, 2019.

Kenji Suzuki. Overview of deep learning in medical imaging. *Radiological physics and technology*, 10(3):257–273, 2017.

Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern recognition*. Elsevier, 2006.

Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

Meet Vadera, Jinyang Li, Adam Cobb, Brian Jalaian, Tarek Abdelzaher, and Benjamin Marlin. Ursabench: A system for comprehensive benchmarking of bayesian deep neural network models and inference methods. *Proceedings of Machine Learning and Systems*, 4:217–237, 2022.

Meet P. Vadera and Benjamin M. Marlin. Challenges and opportunities in approximate bayesian deep learning for intelligent iot systems. In *2021 IEEE Second International Conference on Cognitive Machine Intelligence (CogMI)*, 2021.

Meet P. Vadera, Adam D. Cobb, B Jalaian, and Benjamin M. Marlin. URSABench: Comprehensive Benchmarking of Approximate Bayesian Inference Methods for Deep Neural Networks. In *ICML Workshop on Uncertainty and Robustness in Deep Learning*, 2020a.

Meet P. Vadera, Brian Jalaian, and Benjamin M. Marlin. Generalized bayesian posterior expectation distillation for deep neural networks. In Jonas Peters and David Sontag, editors, *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 124 of *Proceedings of Machine Learning Research*, pages 719–728. PMLR, 03–06 Aug 2020b. URL `https://proceedings.mlr.press/v124/vadera20a.html`.

Meet P. Vadera, Soumya Ghosh, Kenney Ng, and Benjamin M. Marlin. Post-hoc loss-calibration for bayesian neural networks. In *UAI*, 2021.

Haohan Wang, Songwei Ge, Zachary Lipton, and Eric P Xing. Learning robust global representations by penalizing local predictive power. In *Advances in Neural Information Processing Systems*, pages 10506–10518, 2019.

Kuan-Chieh Wang, Paul Vicol, James Lucas, Li Gu, Roger Grosse, and Richard Zemel. Adversarial distillation of bayesian neural network posteriors. *arXiv preprint arXiv:1806.10317*, 2018.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688, 2011.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pages 2074–2082, 2016.

Andrew Gordon Wilson. The case for bayesian deep learning, 2020.

Andrew Gordon Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *ArXiv*, abs/1708.07747, 2017.

Jiayu Yao, Weiwei Pan, S. Ghosh, and Finale Doshi-Velez. Quality of uncertainty quantification for bayesian neural network inference. In *ICML Workshop on Uncertainty and Robustness in Deep Learning*, 2019.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *ArXiv*, abs/1605.07146, 2016.

Ruqi Zhang, Chunyuan Li, Jianyi Zhang, Changyou Chen, and Andrew Gordon Wilson. Cyclical stochastic gradient mcmc for bayesian deep learning. In *ICLR*, 2020.

Yichi Zhang and Zhijian Ou. Learning sparse structured ensembles with stochastic gradient mcmc sampling and network pruning. *2018 IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6, 2018.

Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable DETR: deformable transformers for end-to-end object detection. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL `https://openreview.net/forum?id=gZ9hCDWe6ke`.

C Lawrence Zitnick and Piotr Dollár. Edge boxes: Locating object proposals from edges. In *European conference on computer vision*, pages 391–405. Springer, 2014.