Master Thesis

Uncertainty Quantification in Deep Learning

Master Candidate:
Nathaniel Cogneaux
PSL & Padua Universities

Supervisor:
Pr. Sungyong Baik
Hanyang University

Academic year
2023-2024

This thesis is submitted in partial fulfillment of the requirements for the double Master's degree at University Paris Sciences & Lettres, Paris, France, and the University of Padua, Padua, Italy.

The research was carried out as part of a research internship at Hanyang University, Seoul, South Korea.

Abstract

Deep neural networks (DNNs) have excelled in fields like computer vision, natural language processing, and scientific applications. However, despite their accuracy, DNNs can produce overly confident but incorrect predictions, posing risks in critical areas such as autonomous driving and medical diagnosis.

Uncertainty quantification (UQ) is key to assessing the reliability of predictions beyond accuracy. While state-of-the-art UQ methods are effective, they often require retraining models or running multiple forward passes, making them computationally expensive and impractical for many real-world scenarios. Additionally, most methods cannot be applied to pre-trained models without significant modifications or deep architecture knowledge.

We address this gap with a post-hoc UQ technique that matches state-of-the-art performance while being far more efficient. Our multi-output meta-model enhances pre-trained models with accurate uncertainty estimation, without requiring additional data or retraining. This method drastically reduces computational costs while maintaining the quality of uncertainty quantification, making it both scalable and adaptable to tasks like out-of-domain detection and misclassification.

Our approach delivers near state-of-the-art results on several benchmarks, offering a practical, efficient alternative for real-world applications where accuracy and computational efficiency are crucial.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Professor Sungyong Baik, for the opportunity to conduct research at Hanyang University, Seoul, South Korea. His support, guidance, and expertise were invaluable in shaping this thesis and expanding my knowledge in AI.

I am also grateful to the faculty and staff at Paris Dauphine University and the University of Padua for their remarkable education.

I sincerely thank the Paris Mathematical Sciences Foundation for its generous financial support, which allowed me to focus on my research without financial concerns.

My heartfelt appreciation goes to my family and friends, especially my grandparents for their unwavering support from the South of France. I particularly acknowledge my grandfather, who bravely underwent surgery during the final stages of this thesis, inspiring me with his resilience once again.

Finally, I dedicate this thesis to my parents, whose love, guidance, and support have been my constant source of strength and motivation.

Contents

1 Introduction 1

1.1 Overview . . . 1

1.2 What do we mean by uncertainty? . . . 2

1.3 Different types of uncertainties . . . 4

1.4 Applications . . . 7

1.4.1 Healthcare . . . 7

1.4.2 Self-Driving Cars . . . 8

1.4.3 Conversational Dialog Systems . . . 8

1.4.4 And many others... . . . 8

1.5 Thesis Overview and Contributions . . . 9

2 Basics of Deep Learning and Uncertainty Estimation 11

2.1 Feed-forward neural networks . . . 12

2.1.1 Definition . . . 12

2.1.2 Impact on neural network performance . . . 13

2.1.3 Understanding model overconfidence . . . 15

2.2 Modeling and measuring uncertainty . . . 16

2.2.1 Bayesian approach to modeling aleatoric and epistemic uncertainties . 16

2.2.2 Measuring the quality of uncertainty: calibration error . . . 21

2.2.3 Proper scoring rules . . . 23

3 Comprehensive Literature review 25

3.1 Bayesian techniques . . . 25

3.1.1 Bayesian Neural Networks . . . 25

3.1.2 Sampling Method: Monte Carlo Dropout . . . 30

3.2 Ensembles . . . 32

3.3 Hierarchical Methods - Evidential Deep Learning . . . 34

3.4 Other methods . . . 38
3.5 Post-hoc Single-Pass Uncertainty Quantification methods . . . 39

4 Proposed Method **42**
4.1 Method Description . . . 43
4.2 Uncertainty quantification and disentanglement . . . 46

5 Experiments **49**
5.1 Experimental settings . . . 49
5.2 Uncertainty Quantification Experimental results . . . 50
5.2.1 Cifar-10 . . . 50
5.2.2 Cifar-100 . . . 52
5.2.3 MNIST . . . 54
5.3 OOD Experimental results . . . 55
5.4 Understanding the method . . . 63
5.4.1 The training process and the hyperparameters . . . 63
5.4.2 Trade-off between accuracy and calibration . . . 65

6 Conclusion **67**

List of Figures

1.1 Illustrations of uncertainty in classification and regression (Image credit: Eric Nalisnick) . . . 3
1.2 A schematic view of main differences between aleatoric and epistemic uncertainties from Abdar et al. [2021] . . . 4
1.3 Examples of data uncertainty, class overlap causing an ambiguous decision boundary (Image source: He and Jiang [2024]). . . . 5
1.4 Illustrations of epistemic uncertainty in classification (Image source: Google AI Brain Team) . . . 6
1.5 More detailed schematic view of the two types of uncertainties from He and Jiang [2024] . . . 7
1.6 Example of diabetic retinopathy detection from fundus images. (Image source: Gulshan et al. [2016]). . . . 7

2.1 Illustration of an artificial neuron . . . 12
2.2 Varying intensity of dataset shift in ImageNet-C, including various types of noise and distortions such as Gaussian noise, motion blur, and contrast changes (Image source: Hendrycks and Dietterich [2019]). . . . 14
2.3 Impact of dataset shift on accuracy and uncertainty in neural networks. Accuracy drops with increasing shift intensity Ovadia et al. [2019]. . . . 14
2.4 Impact of dataset shift on the quality of uncertainty in neural networks. Overconfidence increases with increasing shift intensity Ovadia et al. [2019]. . . . 15
2.5 Images that MNIST DNNs believe with 99.99% confidence are digits 0-9. Each column is a digit class (Image source: Nguyen et al. [2015]). . . . 15
2.6 Comparison between poor and proper OOD detection behaviors in deep neural networks. Yellow indicates high uncertainty, the red class corresponds to OOD inputs. (Image source: Liu et al. [2020a]). . . . 16
2.7 Bayesian Linear Regression with an increasing number of points . . . 19

2.8 Evolution of the Posterior distribution with an increasing number of points . 19
2.9 Calibration plots illustrating underconfidence on the left and overconfidence on the right. (Image source: Guo et al. [2017]). 23

3.1 Theoritical representation of Classical Neural Networks vs Bayesian Neural Networks 26
3.2 Comparison of classification boundaries obtained by MAP solutions for LR and BLR. The left image shows the result for LR, and the right image for BLR 28
3.3 Image source: Ulmer et al. [2021] 36
3.4 (Image source: Amini et al. [2020]). 37
3.5 Meta-Model structure (Image source: Shen et al. [2022]). 40

4.1 Number of activated neurons in the penultimate layer (Image source: Ahn et al. [2023]). 42
4.2 Conventional neural networks generate marginal predictions that do not separate true ambiguity from insufficient data. Joint predictions can make this distinction clearer. (Image source: Osband et al. [2023]). 43
4.3 Schematic view of the Multi output module training methodology 45
4.4 Schematic view of the Multi output module inference methodology 46

5.1 Evolution of the outputs and the entropy of the results with an increasing rotation of a 6 on the MNIST dataset with the LeNet5 as a base model . . . 55
5.2 Softmax outputs of the different models in the case of ID (in distribution) data, base model on the left, module in the center and the ensemble on the right, along with the variance of each outputs. 57
5.3 Softmax outputs of the different models in the case of OOD (out of distribution) data, base model on the left, module in the center and the ensemble on the right, along with the variance of each outputs. 58
5.4 Conventional neural networks generate marginal predictions on the left that do not separate joint distribution ambiguity, the module is in the center and the ensemble of base models on the right. 64
5.5 Accuracy with the Number of Heads 65
5.6 ECE with the Number of Heads 65
5.7 Brier Score with the Number of Heads 66
5.8 NLL with the Number of Heads 66

List of Tables

5.1 Cifar-10 Standard results for each model (best result in **bold**, second-best result underlined). 50
5.2 Cifar-10-C Corrupted results for each model (best result in **bold**, second-best result underlined). 51
5.3 Relative improvement: Multi-Output k heads models compared to the Base Model, and Baselines compared to the Deterministic model for Cifar-10 (best result in **bold**, second-best result underlined). 52
5.4 Cifar-100 Standard results for each model (best result in **bold**, second-best result underlined). 52
5.5 Cifar-100-C Corrupted results for each model (best result in **bold**, second-best result underlined). 53
5.6 Relative improvement: Multi-Output k heads models compared to the Base Model, and Baselines compared to the Deterministic model for Cifar-100 (best result in **bold**, second-best result underlined). 53
5.7 MNIST Calibration results for each model (best result in **bold**, second-best result underlined). 54
5.8 Relative improvement: Multi-Output k heads models compared to the Base Model, and Baselines compared to the Deterministic model for MNIST, with number of parameters and forward passes. 55
5.9 OOD Metrics for CIFAR-10 across various datasets and models. Best results are in **bold**, second-best are underlined. For AUR, the best result is the lowest value. 60
5.10 OOD Metrics for CIFAR-100 across various datasets and models. Best results are in **bold**, second-best are underlined. For AUR, the best result is the lowest value. 61

5.11 OOD Metrics for MNIST across various datasets and models. Best results are in **bold**, second-best are underlined. For AUR, the best result is the lowest value. 62
5.12 Number of epochs, training time, and number of parameters for each model across CIFAR-10, CIFAR-100, and MNIST. 64
5.13 Hyperparameters for CIFAR and MNIST datasets. 64

List of Acronyms & Symbols

DNN	Deep Neural Network
SGD	Stochastic Gradient Decent
UQ	Uncertainty Quantification
FCNs	Fully Connected Networks
MLP	Multi-Layer Perceptron
PU	Predictive Uncertainty
EU	Epistemic Uncertainty
AU	Aleatoric Uncertainty
BNN	Bayesian Neural Network
MC	Monte Carlo
DE	Deep Ensemble
VI	Variational Inference
MCMC	Monte Carlo Markov Chains
MH	Metropolis-Hasting
HMC	Hamiltonian Monte Carlo
LV	Latent Variables
iid	independent and identically distributed
ECE	Expected Calibration Error
MCE	Maximum Calibration Error
EDL	Evidential Deep Learning
BS	Brier Score
BMA	Bayesian Model Averaging
GP	Gaussian Process
MIMO	Multi Input Multi Output
MSP	Maximum Softmax Probability

Chapter 1

Introduction

1.1 Overview

Since the early 2010s, the field of machine learning has undergone a profound transformation, largely driven by the resurgence and success of deep learning. Before this era, during the 1990s and 2000s, the training of large deep models was hindered by computational and data limitations. However, the "deep revolution" emerged in 2012 with the groundbreaking success of Convolutional Neural Networks (ConvNets) Krizhevsky et al. [2012], Simonyan and Zisserman [2015], marking a pivotal moment in the field. These advancements simplified data analysis, leading to the widespread adoption of deep learning across various applications, from image classification and speech recognition to chatbots, translation, gaming, and robotics, exemplified by achievements such as AlphaGo.

ConvNets introduced key architectural innovations, such as sparse connectivity and weight sharing through convolutions with learned kernels, allowing for efficient local feature extraction and addressing the parameter explosion problem seen in Fully Connected Networks (FCNs). While Convolutional Neural Networks became a cornerstone in vision tasks, Recurrent Neural Networks (RNNs), including Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs), had already been advancing sequence processing tasks such as speech recognition and text generation. The success of these architectures helped establish neural networks as state-of-the-art across a broad range of domains.

The field saw further breakthroughs with the introduction of Residual Networks (ResNets) in 2015, which tackled the challenges of training very deep networks by employing skip connections to mitigate the vanishing gradient problem He et al. [2016]. This innovation enabled the development of models with hundreds or even thousands of layers, significantly improving performance in image classification and other tasks. These advances built on earlier work, such as the establishment of the ImageNet database Deng et al. [2009], which provided a large-scale benchmark for image classification, and the subsequent development of deep learning techniques LeCun et al. [2015].

Then, in 2017, the emergence of Transformer models, as presented in the seminal paper “Attention is All You Need” Vaswani et al. [2017], revolutionized sequence processing and natural language understanding. Transformers replaced the sequential data processing of RNNs with self-attention mechanisms, enabling parallel processing and capturing long-range dependencies more effectively. This architectural breakthrough has led to state-of-the-art performance in language modeling, translation, and beyond, influencing domains such as image generation and protein structure prediction Pandey et al. [2022].

Despite these advancements, a critical challenge remains in deep learning: robustness and trustworthiness. While deep learning models have achieved substantial gains in average performance metrics such as precision for classification tasks Bengio et al. [2013], their deployment in safety-critical applications necessitates performance certification. This certification involves a formal understanding of the models’ generalization capabilities, optimization landscapes, and, importantly, the stability of their decision functions, in the face of noisy inputs or adversarial attacks Goodfellow et al. [2015].

A crucial aspect of performance certification is the reliable estimation of confidence or uncertainty in the models’ decisions. Traditional deep learning models often exhibit overconfidence Guo et al. [2017], which is inadequate for tasks requiring precise uncertainty estimation, such as healthcare Begoli et al. [2019] and autonomous vehicle navigation Choi et al. [2019].

1.2 What do we mean by uncertainty?

When we talk about uncertainty in machine learning, we refer to returning a *distribution* over predictions rather than a single prediction. This approach provides additional information about how confident the model is in its predictions.

In classification tasks, uncertainty allows us to output a label along with its associated *confidence*. This confidence level indicates how sure the model is about the assigned label.

For regression tasks, uncertainty is expressed in terms of the output *mean* along with its *variance*. The variance captures the model’s uncertainty about the predicted value.

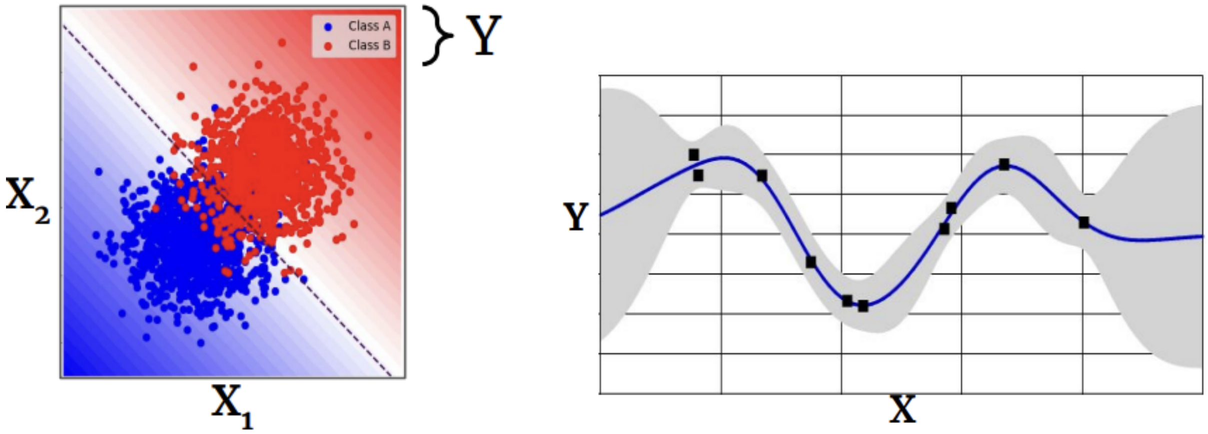


Figure 1.1: Illustrations of uncertainty in classification and regression (Image credit: Eric Nalisnick)

Good uncertainty estimates are crucial because they help us determine when we can trust the model's predictions.

In machine learning, most theoretical models assume that the training and testing data are drawn independently and identically distributed (iid) from the same dataset. This is represented as:

$$\mathbb{P}_{\text{test}}(y, x) = \mathbb{P}_{\text{train}}(y, x)$$

However, in practice, the reality often differs. The test set may come from a different distribution than the training set. This situation is referred to as *out-of-distribution* (OOD) data, where:

$$\mathbb{P}_{\text{test}}(y, x) \neq \mathbb{P}_{\text{train}}(y, x)$$

When the training and test sets follow the same distribution, the model's performance is typically more predictable and aligns with theoretical expectations. In contrast, when the test data distribution differs from the training data distribution, the model's ability to generalize and make accurate predictions is challenged. This is known as *out-of-distribution robustness*, and it is crucial for models to handle such scenarios effectively.

There are different types of so-called *dataset shifts*:

- **Covariate shift**: The distribution of input features x changes, although the conditional distribution of the labels given the features $\mathbb{P}(y|x)$ remains constant. This is commonly seen in scenarios where the nature of the inputs changes over time but the underlying relationship between the inputs and outputs remains stable.

- **Open-set recognition**: New classes that were not present in the training data may appear during testing. For example, a model trained on common animal classes might be tested on exotic animals not seen during training.
- **Label shift**: The distribution of labels $\mathbb{P}(y)$ changes while the conditional distribution of the inputs given the labels $\mathbb{P}(x|y)$ stays the same. This is prevalent in fields like medicine, where the prevalence of diseases may change over time, affecting the label distribution.

These shifts pose significant challenges to maintaining model accuracy and require robust machine learning strategies that can adapt to changes in the input data or environment.

1.3 Different types of uncertainties

"Knowing what a model does not know" comes down to placing appropriate uncertainty scores in its predictions, also called uncertainty quantification (UQ). Uncertainty in DNNs may come from different types of sources, generally classified into two types: aleatoric and epistemic Gal [2016].

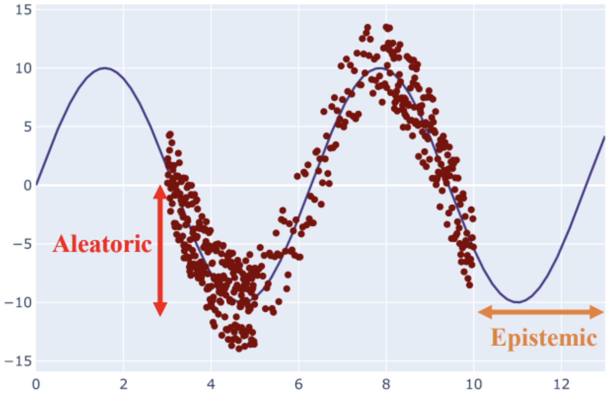


Figure 1.2: A schematic view of main differences between aleatoric and epistemic uncertainties from Abdar et al. [2021]

Aleatoric uncertainty, derived from the Latin word *Aleator*, meaning "dice player", refers to the inherent randomness present in observations. This type of uncertainty captures the natural variability within the data, which can arise due to factors such as stochastic processes, noise in the observations, or inconsistencies in data collection.

For instance, labeling noise can occur due to differences in human judgment, leading to inconsistent labels. In medical imaging, this might manifest as annotators disagreeing on the classification of a particular image, introducing variability in the labels. Measurement noise is another source, where imprecise tools or instruments add variability to the data, such as blurry images caused by poor focus during data collection. Additionally, aleatoric uncertainty is evident in situations involving class overlap, where distinguishing between two or more classes becomes challenging due to their inherent similarities. This overlap can lead to ambiguous data points that are difficult to categorize accurately.

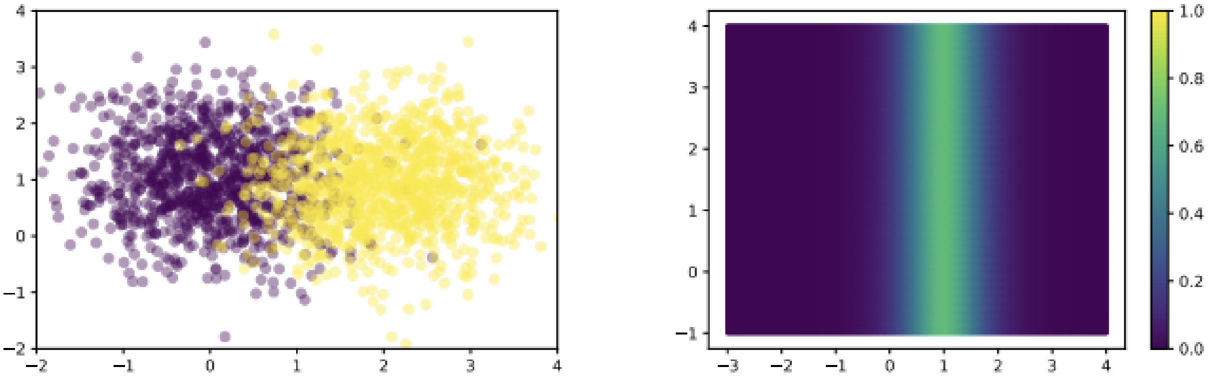


Figure 1.3: Examples of data uncertainty, class overlap causing an ambiguous decision boundary (Image source: He and Jiang [2024]).

Aleatoric uncertainty is considered *irreducible* because it cannot be eliminated simply by gathering more data. This type of uncertainty is an intrinsic property of the data, reflecting the inherent noise, bias and variability that exists within the observations. However, addressing it may require improving the quality of inputs, enhancing the sensors used for data collection, or incorporating additional features and views of the data to provide more context and reduce the impact of noise.

Within aleatoric uncertainty, we can further distinguish between *homoscedastic* and *heteroscedastic* uncertainties. Homoscedastic uncertainty remains constant across different inputs, representing a uniform level of noise throughout the data. In contrast, heteroscedastic uncertainty varies with the input, indicating that some areas of the data space may have more inherent noise than others. This variability can often be learned from the data, allowing models to adjust their confidence levels based on the specific characteristics of each input.

On the other hand, epistemic uncertainty, derived from the Greek word *Episteme*, meaning “knowledge”, reflects the uncertainty inherent in the model itself. This type of uncertainty arises from a lack of knowledge about the best model to explain the observed data.

A key characteristic of epistemic uncertainty is its reducibility. Unlike aleatoric uncertainty, which is intrinsic to the data, epistemic uncertainty can be diminished by acquiring more data, especially in areas where the model's knowledge is sparse. As the dataset grows to infinity, especially with diverse and representative data, epistemic uncertainty should theoretically approach zero, provided the model is well-suited to the task. It occurs because multiple models can fit the training data well, leading to ambiguity about which model to choose. This is particularly relevant when the model has not seen enough data to make definitive predictions. The uncertainty can exist within the same hypothesis class (e.g., different linear classifiers) or across different hypothesis classes (e.g., comparing linear and non-linear models). If the model is uncertain about the appropriate decision boundary, additional data points can help clarify this boundary, leading to a convergence towards a single, well-defined hypothesis. This reduction of epistemic uncertainty is essential for improving model performance, as it underscores the importance of data acquisition in refining model predictions. This type of uncertainty is crucial for out-of-distribution (OOD) detection, as it enables the model to identify unfamiliar samples with a higher uncertainty score.

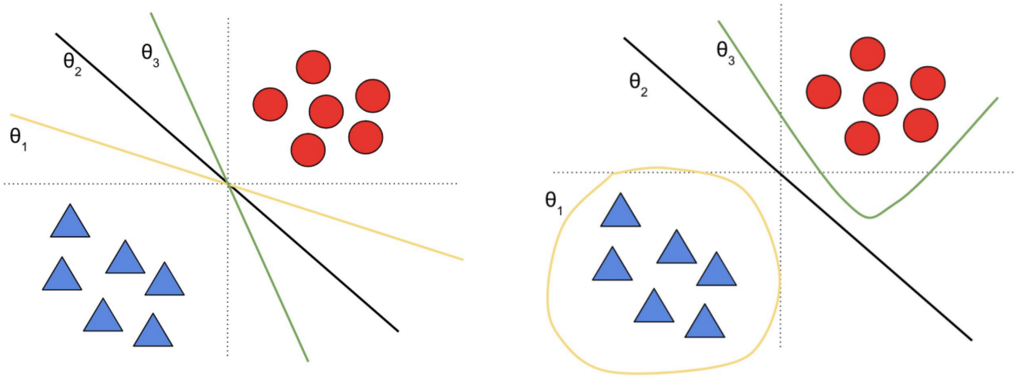


Figure 1.4: Illustrations of epistemic uncertainty in classification (Image source: Google AI Brain Team)

The distinction between aleatoric and epistemic uncertainty is fundamental for developing machine learning models that are not only accurate but also reliable in their predictions. By understanding and quantifying these uncertainties, we can better gauge the confidence of our models in their predictions, enabling safer applications in fields where precision and reliability are crucial.

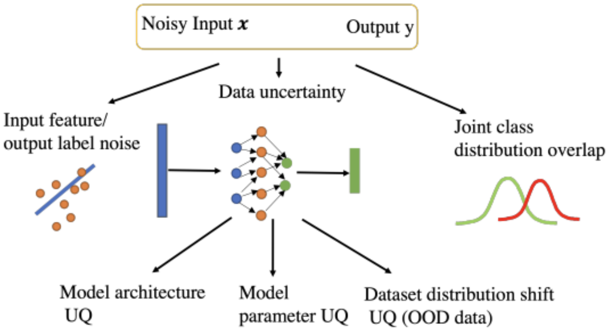

Noisy Input $\boldsymbol{x}$
Output y
Data uncertainty
Input feature/ output label noise
Joint class distribution overlap
Model architecture UQ
Model parameter UQ
Dataset distribution shift UQ (OOD data)

Figure 1.5: More detailed schematic view of the two types of uncertainties from He and Jiang [2024]

1.4 Applications

1.4.1 Healthcare

One of the most critical and rapidly growing applications of uncertainty estimation is in healthcare, particularly in medical imaging and radiology. In medical diagnostics, it's not sufficient to simply classify an image as indicating "disease" or "no disease". Instead, it's crucial to provide a confidence level along with the classification. For example, instead of merely stating that a scan shows a tumor, the model should output that there is an 80% chance that the image indicates a tumor. This additional information is invaluable to doctors, allowing them to make more informed decisions.

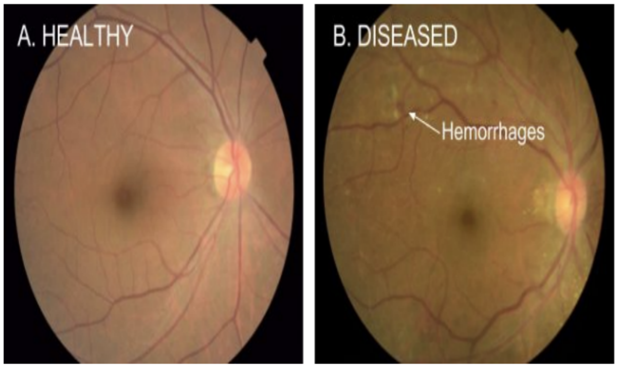


Figure 1.6: Example of diabetic retinopathy detection from fundus images. (Image source: Gulshan et al. [2016]).

The goal is to pass along reliable uncertainty estimates to downstream healthcare professionals. If the model is uncertain, it should have the ability to defer the decision to a human expert. This approach not only improves the safety and reliability of AI systems in healthcare but also ensures that low-quality inputs, which might lead to incorrect predictions, are identified and flagged for further review by human experts.

1.4.2 Self-Driving Cars

Self-driving cars must effectively manage uncertainty, particularly in out-of-distribution (OOD) scenarios caused by dataset shifts Sun et al. [2020]. These shifts can result from factors such as changes in lighting, time of day, weather, or geographical location, all of which impact the car's ability to recognize objects and make decisions. For instance, daylight and nighttime conditions, or urban versus suburban environments, present unique challenges. Temporary factors like road construction or unexpected obstacles further complicate the model's input.

When the model encounters significant deviations from its training data, such as snow-covered roads or other OOD examples, it should increase its uncertainty, enabling the system to take precautionary actions or hand over control to the driver, ensuring safety in unpredictable conditions.

1.4.3 Conversational Dialog Systems

In the realm of conversational dialog systems, such as voice assistants, handling out-of-scope (OOS) utterances is a critical challenge. These systems must be capable of detecting when they do not understand a user's request, expressing uncertainty, and responding appropriately. A well-designed conversational system should avoid providing random or potentially misleading answers when it does not fully comprehend the user's query. Instead, it should recognize that the query is out-of-scope and either ask the user to repeat the question or provide a fallback response.

For example, if a user asks a question outside the system's domain of knowledge—such as a finance-related system being asked about sports—the system should express uncertainty. Rather than offering an incorrect answer, it could request clarification or acknowledge its inability to respond to the query. This approach enhances the user experience by preventing the system from delivering irrelevant or confusing responses Larson et al. [2019].

1.4.4 And many others...

In decision-making processes, reliable confidence estimates are fundamental, ensuring that the decisions made are based on trustworthy information. In active learning and lifelong learning, focusing on uncertain predictions during training helps to continuously improve model performance, enabling the model to learn more effectively from data. In reinforcement learning for instance, uncertainty helps balance exploration and exploitation, guiding the learning process to optimize performance Depeweg et al. [2018].

1.5 Thesis Overview and Contributions

The thesis begins by introducing the foundational concepts of deep neural networks (DNNs) and their architecture, followed by the significance of Uncertainty Quantification (UQ) in machine learning. It explains how uncertainties are modeled and introduces common UQ metrics, which are essential for assessing model confidence.

Next, the thesis provides a comprehensive literature review of existing UQ methods, starting with fundamental approaches like Bayesian methods, which incorporate uncertainty directly into the model, and ensemble methods, which estimate uncertainty through multiple models. Emerging techniques, such as Evidential Deep Learning (EDL), are also explored. The review then moves to post-hoc methods, which introduce UQ into pre-trained models without retraining or significant architectural modifications. The strengths and limitations of these methods are assessed, with a focus on computational efficiency and practicality. While many current UQ methods perform well, they are often computationally intensive, requiring retraining or multiple forward passes, or demand deep knowledge of the model's architecture, making them less adaptable. This highlights a clear research gap: the need for a more efficient, model-agnostic solution that works with existing models while preserving high accuracy and reliability. The proposed method addresses this gap by being both model-agnostic and potentially task-agnostic, making it applicable to a wide range of supervised learning problems.

Specifically, we introduce a novel multi-output module architecture designed to efficiently add UQ capabilities to pre-trained models. This method enables multiple independent predictions in a single forward pass, without requiring additional data or retraining the model. It also incorporates a new approach to disentangling uncertainties by leveraging the disagreement between outputs and using overlapping data to improve prediction robustness. The proposed solution is computationally efficient and integrates well with standard UQ metrics, making it both scalable and practical.

The experimental evaluation validates the effectiveness of the proposed method on benchmark datasets such as MNIST, CIFAR-10, and CIFAR-100, including their corrupted variants.

The results demonstrate that the multi-output meta-model achieves near state-of-the-art performance while significantly reducing computational costs. A detailed comparison with existing UQ techniques highlights the method's efficiency, delivering competitive uncertainty estimates in a fraction of the time.

Chapter 2

Basics of Deep Learning and Uncertainty Estimation

The concept of neural networks has evolved significantly over the past eight decades. Initially designed to mimic the behavior of real neurons, these networks have developed into complex architectures with multiple sub-modules. An early example is the Multi-Layer Perceptron (MLP), introduced by Rumelhart, Hinton, and Williams in 1985, which laid the groundwork for modern deep learning Rumelhart et al. [1985]. Today, sophisticated architectures like AlphaFold Jumper et al. [2021] exemplify the advanced networks capable of solving intricate tasks such as protein structure prediction. Despite these advancements, the underlying structure of neural networks remains consistent. As Yann LeCun succinctly defines it, "Deep Learning is constructing networks of parameterized functional modules and training them from examples using gradient-based optimization".

Deep learning involves training models on data, which consists of *features* or *inputs* the variables describing each example and, in *supervised learning*, *labels*, the correct outputs associated with those inputs. In supervised learning, models are trained on labeled data, where each input is paired with a label, enabling the model to learn the relationship between them. Supervised learning can be categorized into *classification*, where the task is to predict discrete labels, such as identifying spam emails Bishop [2006], and *regression*, where the goal is to predict continuous values, like stock prices Jerome Friedman and Tibshirani [2001]. In contrast, *unsupervised learning* trains models on data without labels, aiming to uncover patterns or structures, such as clustering customers by purchasing behavior Trevor Hastie and Friedman [2009]. Additional paradigms include *semi-supervised learning*, which leverages a small amount of labeled data alongside a larger set of unlabeled data Zhu [2005], and *reinforcement learning*, where an agent learns to make decisions by interacting with an environment to maximize rewards Sutton and Barto [2018].

2.1 Feed-forward neural networks

2.1.1 Definition

At the core of a standard feed-forward neural network is the concept of a neuron: a computational unit that receives inputs, applies a linear transformation using weights and biases, and passes the result through a non-linear activation function. This model is generalized in neural networks, where neurons are organized into layers. In a network with multiple hidden layers, each layer transforms its input x through a series of linear maps $W_1, W_2, \ldots, W_L$ and biases $b_1, b_2, \ldots, b_L$, followed by a non-linear activation function $\sigma(\cdot)$, such as ReLU, allowing the network to capture and learn complex patterns.

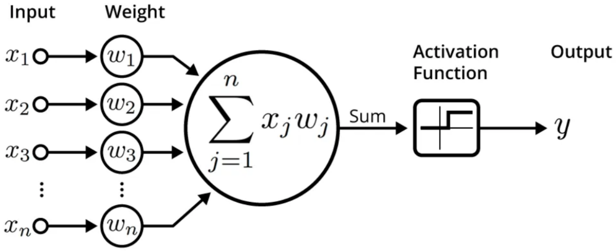


Figure 2.1: Illustration of an artificial neuron

ReLU is defined as $\mathrm{ReLU}(z) = \max(0, z)$ and is widely used due to its simplicity and its ability to mitigate the vanishing gradient problem, though it can suffer from issues like dead neurons Hochreiter and Schmidhuber [1997]. Other common activation functions include the sigmoid function, $\mathrm{sigmoid}(z) = \frac{1}{1+\exp(-z)}$, which maps inputs between 0 and 1, making it useful for binary classification tasks. Another example is the hyperbolic tangent function (tanh), $\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$, which maps inputs between -1 and 1 and is often used when data is centered around zero. Finally, Leaky ReLU, defined as Leaky $\mathrm{ReLU}(z) = \max(\alpha z, z)$ where α is a small constant, is a variation of ReLU that allows for a small gradient when the input is negative, addressing some limitations of standard ReLU.

After passing through the hidden layers, another linear function W_{L+1} can be used to map the final hidden layer to the output:

$$\hat{y} = \sigma(W_L(\sigma(\ldots\sigma(W_1x + b_1))\ldots) + b_L)W_{L+1} \tag{1}$$

Throughout this study, we focus on the supervised learning setting, where labels are provided to train the model. In classification tasks, where the goal is to predict which category X belongs to among a set of C possible classes, the model output $\hat{y}$ is typically passed through a softmax function to compute the normalized probabilities $\hat{p}_d = \frac{\exp(\hat{y}_d)}{\sum_{d'} \exp(\hat{y}_{d'})}$. The final layer in this case consists of C units, each corresponding to a class. The most commonly used loss function for classification is the cross-entropy loss, which quantifies the difference between the predicted probabilities and the true labels:

$$\mathcal{L}^{W_1,\ldots,W_{L+1},b_1,\ldots,b_L}(X,Y) = \mathcal{L}(X,Y) = -\frac{1}{N}\sum_{i=1}^{N}\log(\hat{p}_{i,c_i})$$

where $X = (x_1, \ldots, x_N)$ represents the input data and $Y = (y_1, \ldots, y_N)$ denotes the corresponding labels. Although cross-entropy is the standard choice for classification, alternative loss functions can be applied depending on the task.

For regression tasks, where the objective is to predict a continuous value, the Euclidean loss (or mean squared error) is typically employed:

$$\mathcal{L}(X,Y) = \frac{1}{2N}\sum_{i=1}^{N}\|y_i - \hat{y}_i\|^2$$

In this scenario, the final layer generally contains a single unit to generate the continuous prediction Fisher [1936].

2.1.2 Impact on neural network performance

A practical example of dataset shift can be observed in the ImageNet-C benchmark. In this scenario, the iid test set comprises clean images from the ImageNet dataset. However, various forms of noise and distortions are introduced, creating a dataset shift that makes it increasingly difficult for neural networks to maintain their performance. As shown in Figure 2.2, the shift is simulated by progressively adding noise to the images, with the severity of the corruption increasing from 1 to 5. The clean image represents the iid condition, and as the dataset shift intensifies, the neural network's performance typically degrades.

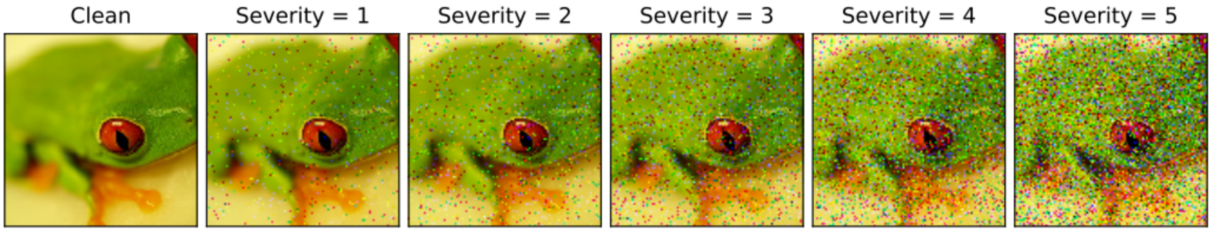


Figure 2.2: Varying intensity of dataset shift in ImageNet-C, including various types of noise and distortions such as Gaussian noise, motion blur, and contrast changes (Image source: Hendrycks and Dietterich [2019]).

This example highlights how neural networks, which perform well on clean, in distribution (ID) data, often struggle with OOD examples. The introduced noise and perturbations lead to significant drops in accuracy, demonstrating the importance of developing models that are robust to such shifts.

As the covariate shift increases, the accuracy of a standard neural network such as ResNet drops significantly. This is evident in the accuracy plot shown in Figure 2.4, where the accuracy decreases progressively as the severity of the dataset shift intensifies from clean images to those with higher noise levels. Ideally, a model should be able to recognize when its predictions are less certain, especially as accuracy declines. However, as the shift intensity increases, not only does the accuracy degrade, but the quality of uncertainty estimation also worsens. This is problematic because the model continues to make overconfident predictions despite its declining performance. Such behavior is concerning because it leads to a false sense of confidence in incorrect predictions.

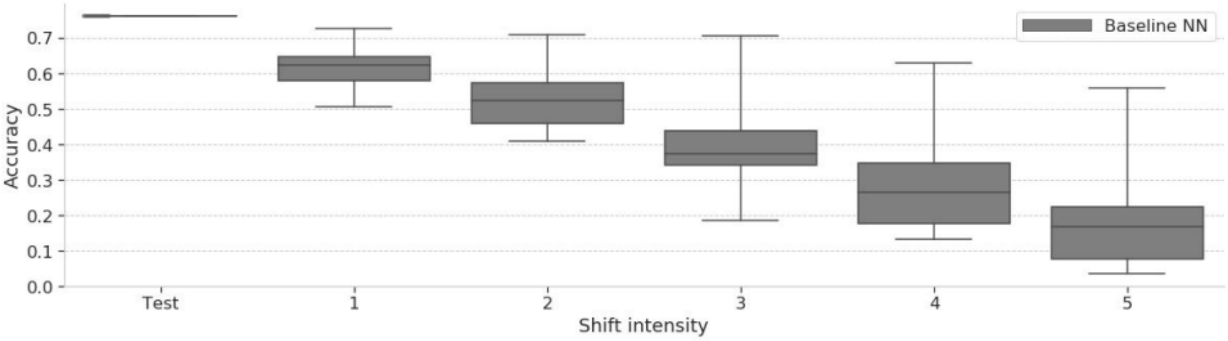


Figure 2.3: Impact of dataset shift on accuracy and uncertainty in neural networks. Accuracy drops with increasing shift intensity Ovadia et al. [2019].

This overconfidence in the face of meaningless data highlights a critical vulnerability in

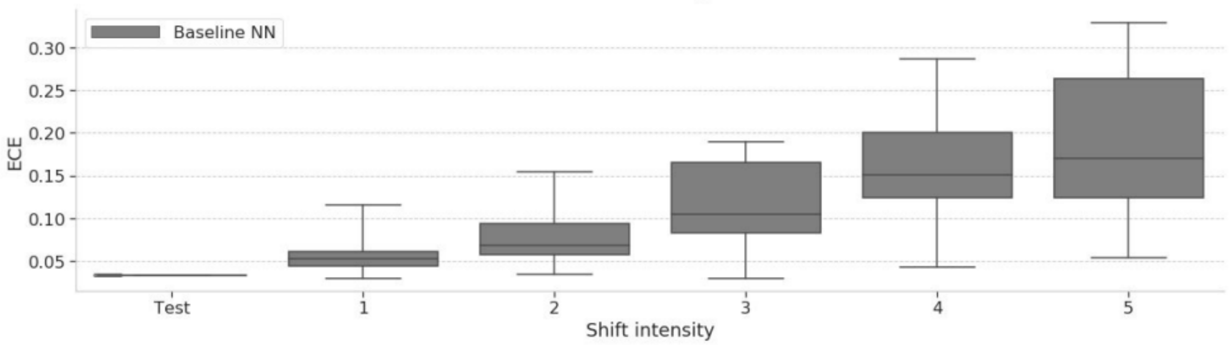


Figure 2.4: Impact of dataset shift on the quality of uncertainty in neural networks. Overconfidence increases with increasing shift intensity Ovadia et al. [2019].

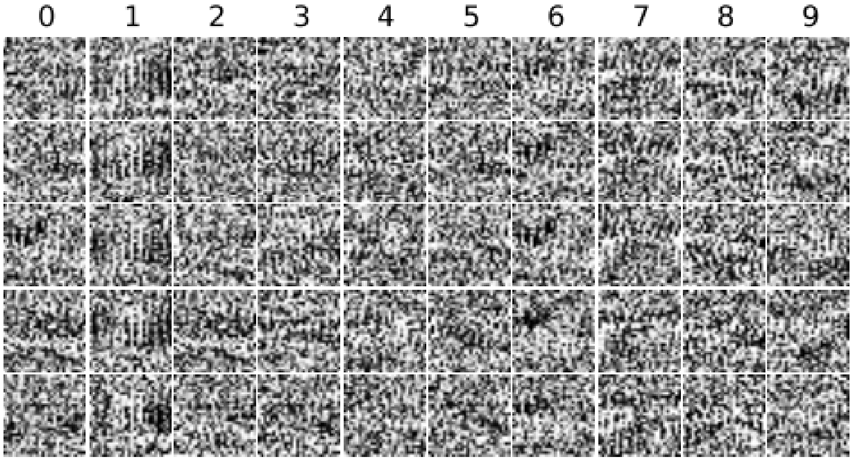


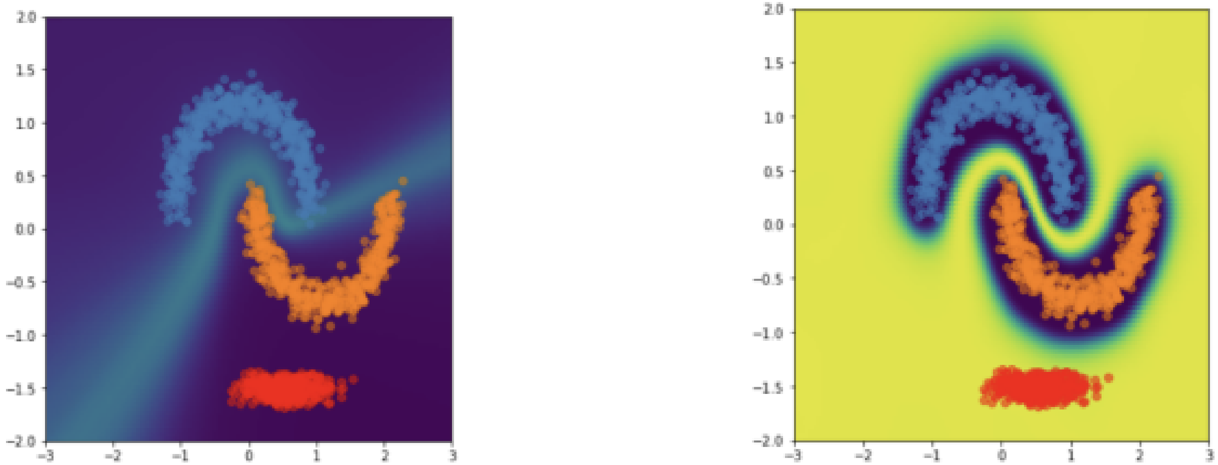
Figure 2.5: Images that MNIST DNNs believe with 99.99% confidence are digits 0-9. Each column is a digit class (Image source: Nguyen et al. [2015]).

neural network models, emphasizing the need for better methods to handle OOD inputs and to calibrate the confidence of predictions appropriately. This example highlights a critical challenge in neural network design: ensuring that the model not only maintains accuracy under distribution shifts but also provides reliable uncertainty estimates, acknowledging when it does not know.

2.1.3 Understanding model overconfidence

In practice, deep neural networks models are often designed as decision boundary classifiers. Instead of becoming uncertain as inputs move away from the training data, the models become more confident in their predictions. The further an input is from the decision boundary, the more certain the model becomes about assigning it to a specific class, as shown in the

2.6. This behavior is problematic because when a model encounters a completely unfamiliar class that it has never seen during training, it fails to acknowledge its uncertainty. Instead of indicating “I don’t know”, the model incorrectly assigns high confidence to the unfamiliar class, treating it as more similar to one of the known classes.



(a) Poor OOD detection by a deep neural network.

(b) Proper OOD detection by an ideal model.

Figure 2.6: Comparison between poor and proper OOD detection behaviors in deep neural networks. Yellow indicates high uncertainty, the red class corresponds to OOD inputs. (Image source: Liu et al. [2020a]).

2.2 Modeling and measuring uncertainty

As discussed earlier, there are two primary types of uncertainty: epistemic (model uncertainty) and aleatoric (data uncertainty) Kiureghian and Ditlevsen [2009]. We define the overall predictive uncertainty (PU) that is composed of the two components: (i) epistemic uncertainty (EU) and (ii) aleatoric uncertainty (AU), and can be typically expressed as their sum:

$$PU = EU + AU \tag{2.1}$$

2.2.1 Bayesian approach to modeling aleatoric and epistemic uncertainties

With the basic architecture of feed-forward neural networks now established, we can proceed to define uncertainties in a mathematical framework. To understand how to extract a good notion of uncertainty in neural networks, it's essential to start with the fundamentals of how these models are typically optimized. Neural networks are usually optimized using *Stochastic Gradient Descent (SGD).* The goal of this optimization is to find the best set of parameters ω that maximize the probability of the data given the model.

$$\omega^* = \arg\max_{\omega} \mathbb{P}(\omega \mid \mathbf{x}, \mathbf{y})$$

This is equivalent to minimizing the negative log-likelihood of the data while also incorporating a prior distribution on the parameters:

$$= \arg\min_{\omega} - \log \mathbb{P}(\mathbf{y} \mid \mathbf{x}, \omega) - \log \mathbb{P}(\omega)$$

While this approach effectively finds a good set of parameters, it yields only a single prediction per example, thereby failing to capture *model uncertainty.* In other words, it produces a single output $\mathbf{y}$ for a given input $\mathbf{x}$, lacking the capacity to express uncertainty about the model parameters.

This section will explore the foundational Bayesian approach to modeling uncertainties, with a focus on how epistemic uncertainty can be represented as a probability distribution over the model's parameters. A more detailed literature review on these concepts will be provided in the subsequent section.

Let $D_{tr} = \{X, Y\} = \{(x_i, y_i)\}_{i=1}^{N}$ denote a training dataset where $x_i \in \mathbb{R}^D$ are the inputs and $y_i \in \{1, \ldots, C\}$ are the corresponding classes, with C representing the number of classes. The goal is to optimize the parameters ω of a function $y = f^{\omega}(x)$ that can predict the desired output. Using a Bayesian approach, we define the model likelihood $\mathbb{P}(y|x, \omega)$. For classification tasks, the likelihood can be modeled using the softmax function:

$$\mathbb{P}(y = c|x, \omega) = \frac{\exp(f_c^{\omega}(x))}{\sum_{c'} \exp(f_{c'}^{\omega}(x))}$$

For regression tasks, a Gaussian likelihood is typically assumed:

$$\mathbb{P}(y|x, \omega) = \mathcal{N}(y; f^{\omega}(x), \tau^{-1} I)$$

where τ represents the model precision. The posterior distribution, $\mathbb{P}(\omega|X, Y)$, for a given dataset D_{tr} over ω can be obtained by applying Bayes' theorem, and is expressed as follows:

$$\mathbb{P}(\omega|X, Y) = \frac{\mathbb{P}(Y|X, \omega)\mathbb{P}(\omega)}{\mathbb{P}(Y|X)}.$$

Then, for a given test sample x^*, the class label with respect to $\mathbb{P}(\omega|X,Y)$ can be predicted by:

$$\mathbb{P}(y^*|x^*,X,Y) = \int \mathbb{P}(y^*|x^*,\omega)\mathbb{P}(\omega|X,Y)d\omega.$$

This process is known as inference or marginalization. We can analytically observe that these two notions resurface when formulating the posterior predictive distribution for a new data point Ulmer et al. [2021]:

$$\mathbb{P}(y^*|x^*,X,Y) = \int \underbrace{\mathbb{P}(y^*|x^*,\omega)}_{\text{Aleatoric}}\underbrace{\mathbb{P}(\omega|X,Y)}_{\text{Epistemic}}d\omega. \tag{2.2}$$

Here, the first factor captures the aleatoric uncertainty regarding the correct prediction, while the second factor expresses uncertainty about the correct model parameters.

Remark: As more data is observed, the density of $\mathbb{P}(\omega|X,Y)$ should increasingly concentrate on reasonable parameter values for ω. To understand this better, here is a general result in Bayesian inference Box and Tiao [1973], we consider the Gaussian distributions for x and y, assuming:

- $\mathbb{P}(x) = \mathcal{N}(x|\mu_x,\Sigma_x)$
- $\mathbb{P}(y|x) = \mathcal{N}(y|Ax+b,\Sigma_y)$

Consequently, the posterior distribution $\mathbb{P}(\omega|X,Y)$ is given by a Gaussian distribution:

$$\mathbb{P}(\omega|X,Y) = \mathcal{N}(\omega;\mu,\Sigma).$$

With:

$$\begin{aligned}\Sigma^{-1} &= \Sigma_x^{-1} + A^T\Sigma_y^{-1}A\\ \mu &= \Sigma[A^T\Sigma_y^{-1}(y-b) + \Sigma_x^{-1}\mu_x]\end{aligned}$$

This Gaussian representation of the posterior provides a comprehensive framework for understanding how our beliefs about the parameters change with the observation of new data. The process unfolds as follows:

- No observation: before any data are observed, the posterior distribution $\mathbb{P}(\omega|X,Y)$ is identical to the prior distribution, indicating our initial beliefs about ω, in other words: $\mathbb{P}(\omega|X,Y) = \mathbb{P}(\omega)$

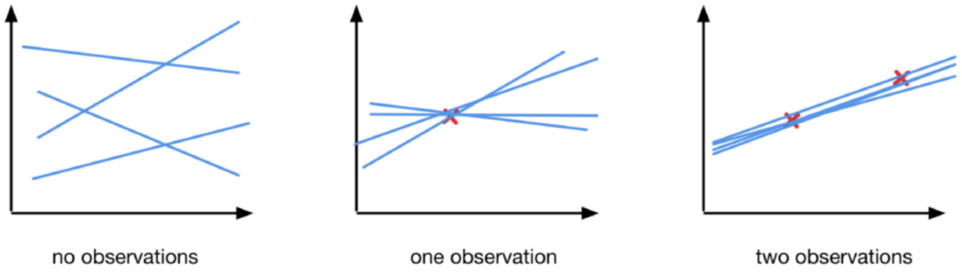


Figure 2.7: Bayesian Linear Regression with an increasing number of points

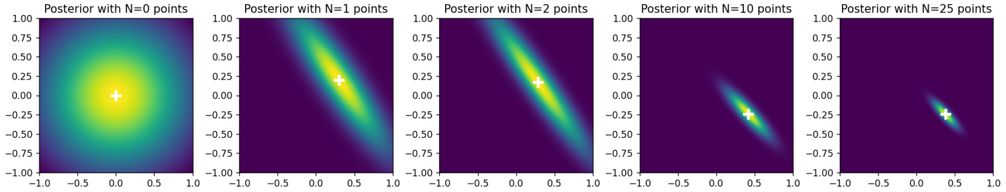


Figure 2.8: Evolution of the Posterior distribution with an increasing number of points

- First Observation: Upon observing the first data point (x_0, y_0), we update our prior belief to reflect this new information through a Bayesian update:

  $$\mathbb{P}(\omega|x_0, y_0) \propto \mathbb{P}(\omega)\mathbb{P}(y_0|x_0, \omega),$$

  where $\mathbb{P}(y_0|x_0, \omega)$ is the likelihood of observing y_0 given x_0 and parameter ω, often assumed to be Gaussian as well.

- Subsequent Observations: Each new observation (x_i, y_i) further refines our posterior belief about ω, integrating the information from all previous observations and the prior:

  $$\mathbb{P}(\omega|(x_0, y_0), \ldots, (x_i, y_i)) \propto \mathbb{P}(\omega) \prod_{j=0}^{i} \mathbb{P}(y_j|x_j, \omega).$$

  This formula demonstrates the cumulative nature of Bayesian updating, where each new data point contributes to an increasingly accurate and nuanced understanding of ω.

We now describe two decompositions of the form 2.1, each differing in the metric used to quantify uncertainty: the first one is based on entropy, while the second one uses variance Mohan et al. [2022], Depeweg et al. [2018].

Let $\mathbb{H}(\cdot)$ compute the differential entropy of a probability distribution. The total uncertainty present in Eq. (3.3) can be quantified as $\mathbb{H}(y^*|x^*, D_{tr})$, with $D_{tr} = \{X, Y\} = \{(x_i, y_i)\}_{i=1}^N$. Precisely, it is defined as:

$$\mathbb{H}(y^*|x^*, D_{tr}) = -\sum_c \mathbb{P}(y^* = c|x^*, \omega) \log \mathbb{P}(y^* = c|x^*, \omega),$$

which can be approximated using Monte Carlo (MC) samples as:

$$\mathbb{H}(y^*|x^*, D_{tr}) = -\sum_c \left(\frac{1}{N} \sum_{i=1}^N \mathbb{P}\left(y^* = c|x^*, \omega^{(i)}\right) \right) \log \left(\frac{1}{N} \sum_{i=1}^N \mathbb{P}\left(y^* = c|x^*, \omega^{(i)}\right) \right)$$

following Gal and Ghahramani [2016].

We use the natural logarithm for all the equations described in this section, and the values are reported in nats, which is the natural unit of information.

Furthermore, assume that we do not integrate ω out in Eq. (3.3) and, instead, we just condition on a specific value of this variable. Then the corresponding uncertainty is just $\mathbb{H}(y^*|\omega, x^*)$ and it corresponds to the aleatoric uncertainty given the *fixed* parameter ω. The expectation of this quantity under the posterior, $\mathbb{E}_{\mathbb{P}(\omega|D_{tr})}[\mathbb{H}(y^*|\omega, x^*)]$, can then be used to quantify the overall aleatoric uncertainty in Eq. (3.3). As a consequence, we can quantify the epistemic part of the uncertainty in Eq. (3.3) by computing the difference between total and aleatoric uncertainties:

$$\mathbb{H}[y^*|x^*, D_{tr}] - \mathbb{E}_{\mathbb{P}(\omega|D_{tr})}[\mathbb{H}(y^*|\omega, x^*)] = I(y^*, \omega|x^*, D_{tr}), \tag{2.3}$$

which is the mutual information between y^* and ω that can also be approximated as

$$\begin{aligned} \mathbb{I}(y, \omega|x, D_{tr}) = &-\sum_c \left(\frac{1}{N} \sum_{i=1}^N \mathbb{P}\left(y = c|x, \omega^{(i)}\right) \right) \log \left(\frac{1}{N} \sum_{i=1}^N \mathbb{P}\left(y = c|x, \omega^{(i)}\right) \right) \\ &+ \frac{1}{N} \sum_{c,N} \sum_{i=1}^N \mathbb{P}\left(y = c|x, \omega^{(i)}\right) \log \mathbb{P}\left(y = c|x, \omega^{(i)}\right) \end{aligned}$$

following Gal and Ghahramani [2016].

Instead of entropy, we can also use variance as a measure of uncertainty Depeweg et al. [2018]. Let $\sigma^2(\cdot)$ compute the variance of a probability distribution. The total uncertainty present in Eq. (3.3) is then $\sigma^2(y^*|x^*, D_{tr})$. This quantity can then be decomposed using the law of total variance:

$$\sigma^2(y^*|x^*, D_{tr}) = \sigma^2_{\mathbb{P}(\omega|D_{tr})}(\mathbb{E}[y^*|\omega, x^*]) + \mathbb{E}_{\mathbb{P}(\omega|D_{tr})}[\sigma^2(y^*|\omega, x^*)]. \tag{2.4}$$

where $\mathbb{E}[y^*|\omega, x^*]$ and $\sigma^2[y^*|\omega, x^*]$ are, respectively, the mean and variance of y^* according to $\mathbb{P}(y^*|\omega, x^*)$. In the expression above, $\sigma^2_{\mathbb{P}(\omega|D_{tr})}(\mathbb{E}[y^*|\omega, x^*])$ is the variance of $\mathbb{E}[y^*|\omega, x^*]$ when $\omega \sim \mathbb{P}(\omega|D_{tr})$. This term only considers the effect of ω. Therefore, it corresponds to the epistemic uncertainty in Eq. (3.3). By contrast, the term $\mathbb{E}_{\mathbb{P}(\omega|D_{tr})}[\sigma^2(y^*|\omega, x^*)]$ represents the average value of $\sigma^2(y^*|\omega, x^*)$ when $\omega \sim \mathbb{P}(\omega|D_{tr})$. This term ignores any contribution to the variance of y^* from ω and, therefore, it represents the aleatoric uncertainty in Eq. (3.3).

In practice, however, the posterior distribution over weights in neural networks is intractable, and approximating it often incurs significant computational costs Neal [1996]. This intractability arises because the marginal likelihood $\mathbb{P}(Y|X)$, which is necessary for computing the posterior $\mathbb{P}(\omega|X, Y) = \frac{\mathbb{P}(Y|X,\omega)\mathbb{P}(\omega)}{\mathbb{P}(Y|X)}$, cannot be evaluated analytically. As a result, numerous models and approximation techniques have been developed by researchers to perform uncertainty quantification (UQ) while circumventing this challenge.

2.2.2 Measuring the quality of uncertainty: calibration error

A popular method for assessing the quality of uncertainty estimates in machine learning models is the *calibration error*. Calibration measures how well the predicted confidence levels align with the actual observed accuracy.

The calibration error can be defined as the absolute difference between confidence and accuracy:

$$\text{Calibration Error} = |\text{Confidence} - \text{Accuracy}|$$

This metric helps to evaluate whether a model's predicted probabilities are well-calibrated with the real-world outcomes.

Example: Consider a weather forecasting model that predicts a 80% chance of rain. If, on all the days where this prediction was made, it actually rained 80% of the time, the model is said to be perfectly calibrated.

In regression tasks, calibration corresponds to how often the true values fall within the predicted confidence intervals. For example, if a model predicts that the temperature will be between 14°C and 18°C with 90% confidence, then, ideally, the actual temperature should fall within this range 90% of the time. In other words, calibration in regression is about the coverage of the true values within the predicted confidence intervals, ensuring that the intervals accurately reflect the uncertainty in the predictions.

Now, to evaluate the quality of uncertainty estimates in machine learning, a widely used metric is the *Expected Calibration Error (ECE).* The Expected Calibration Error (ECE) is computed as follows:

$$\mathrm{ECE} = \sum_{b=1}^{B} \frac{n_b}{N} \left| \mathrm{acc}(b) - \mathrm{conf}(b) \right|$$

Where:

- B is the number of bins.
- n_b is the number of samples in bin b.
- N is the total number of samples.
- $\mathrm{acc}(b)$ is the accuracy of the samples in bin b.
- $\mathrm{conf}(b)$ is the average predicted confidence of the samples in bin b.

The process begins by binning the predicted probabilities into B equal-width bins, each representing a range of confidence levels. For each bin, the actual accuracy and the average predicted confidence of the predictions within that bin are calculated. The calibration error for each bin is determined as the absolute difference between the within-bin accuracy and the within-bin confidence. The ECE is then obtained as the weighted average of these errors, with weights proportional to the number of samples in each bin.

The ECE provides an overall measure of the model's calibration across different confidence levels. A lower ECE indicates that the model's predicted confidence closely aligns with its actual accuracy, which is the desired outcome Pakdaman Naeini et al. [2015]. It also helps in diagnosing whether a model tends to be underconfident or overconfident. When the model's confidence is consistently lower than its accuracy (Confidence $<$ Accuracy), the model is said to be underconfident. This means the model is overly cautious, often predicting lower confidence levels than are justified. Conversely, if the model's confidence is higher than its accuracy (Confidence $>$ Accuracy), the model is considered overconfident. This indicates that the model is too sure of itself, often predicting with high confidence even when it is incorrect.

Of course, other metrics can be thought about, for instance, in high-risk applications where reliable confidence measures are absolutely necessary, we may wish to minimize the worst-case

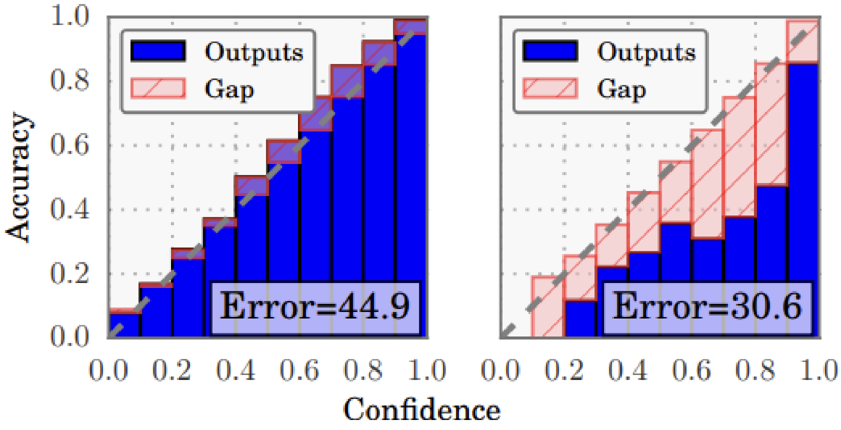


Figure 2.9: Calibration plots illustrating underconfidence on the left and overconfidence on the right. (Image source: Guo et al. [2017]).

deviation between confidence and accuracy:

$$\max_{p \in [0,1]} \left| \mathbb{P}\left(\hat{Y} = Y \mid \hat{P} = p\right) - p \right| .$$

The Maximum Calibration Error (MCE) Pakdaman Naeini et al. [2015] estimates this deviation. Similarly to ECE, this approximation involves binning:

$$\text{MCE} = \max_{b \in \{1,\ldots,B\}} \left| \text{acc}(b) - \text{conf}(b) \right| .$$

On reliability diagrams 3.4. MCE is the largest calibration gap (red bars) across all bins, whereas ECE is a weighted average of all gaps.

2.2.3 Proper scoring rules

While calibration metrics like ECE and MCE provide useful insights into how well the predicted probabilities align with actual outcomes, they do not account for accuracy. A model could, in theory, be perfectly calibrated but still perform poorly if its predictions are consistently random. To address this, we look at *proper scoring rules*, which provides a balanced approach to evaluating uncertainty by incorporating both the model's calibration and its accuracy as well.

Negative Log-Likelihood (NLL) is one of the most commonly used proper scoring rules, also known as cross-entropy. NLL evaluates the likelihood of the true outcomes under the predicted probability distribution. Cross-entropy is widely used as a loss function in classification tasks; however, one downside is that it can overemphasize tail probabilities, leading to models that are overly confident in certain predictions.

Another proper scoring rule is the **Brier Score (BS)**, which provides a quadratic penalty for the difference between predicted probabilities and the actual outcomes Gneiting and Raftery [2007]. The Brier Score is defined as:

$$\mathrm{BS} = \frac{1}{|\mathcal{Y}|} \sum_{y \in \mathcal{Y}} \left[\mathbb{P}(y|\mathbf{x}_n, \omega) - \delta(y - y_n)\right]^2$$

where $|\mathcal{Y}|$ is the number of possible outcomes (classes), $\mathbb{P}(y|\mathbf{x}_n, \omega)$ is the predicted probability of class y given the input $\mathbf{x}_n$ and model parameters ω, and $\delta(y - y_n)$ is the Kronecker delta, which is 1 if $y = y_n$ and 0 otherwise.

A downside of the he Brier Score is that, since it has a bounded range [0,1], unlike logarithmic scoring rules, it can be numerically unstable to optimize, especially when dealing with small probabilities.

Chapter 3

Comprehensive Literature review

In this section, we provide a comprehensive review of the literature on uncertainty estimation. We summarize the characteristics of various methods, highlighting their technical approaches, and compare their advantages and disadvantages in addressing different types of uncertainty. We also assess their efficiency in terms of computational time and memory usage. Finally, we identify the research gaps that we aim to address in the subsequent section.

3.1 Bayesian techniques

Unlike Bayesian Linear Regression, as discussed in 2.2.1, the complexity of many models often prevents closed-form solutions for these distributions. Even a straightforward model like Bayesian Logistic Regression, that deals with the classification problem from a probabilistic standpoint, lacks a Gaussian likelihood and consequently does not have a closed-form solution. To address this, a branch of research focuses on approximating them using Bayesian Neural Networks.

3.1.1 Bayesian Neural Networks

Bayesian Neural Networks (BNNs) incorporate uncertainty by placing probability distributions over the network's weights. By specifying a distribution, BNNs capture a range of plausible models that fit the data, with predictions reflecting uncertainty in regions with sparse data. Unlike traditional neural networks with fixed weights, BNNs compute outputs based on the probability of weights given the inputs and data, $\mathbb{P}(y_i|x_i, D)$, allowing for more robust inference in scenarios with limited or noisy data. The prior distribution over the

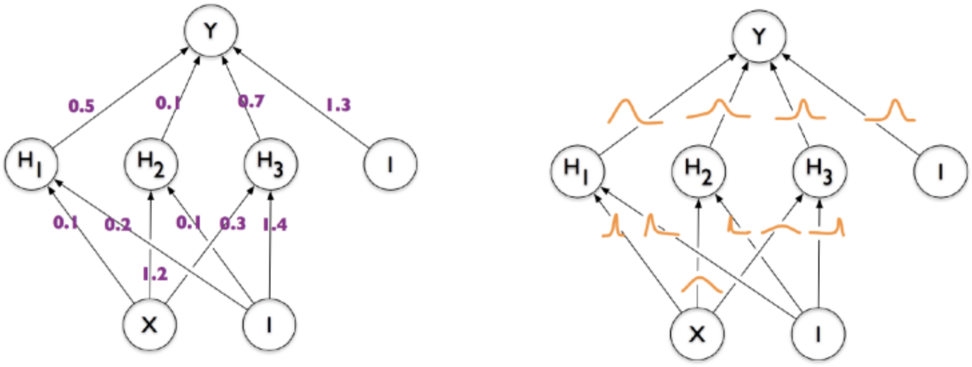


Figure 3.1: Theoritical representation of Classical Neural Networks vs Bayesian Neural Networks

weights is defined as $\mathbb{P}(\omega)$, such as a Gaussian distribution $\mathbb{P}(\omega) = \mathcal{N}(\omega|0, \alpha^{-1}I)$, which reflects the belief that, prior to observing data, weights are likely to be small in magnitude. Alternatively, other choices like the Laplace distribution are also possible. Depending on the choice of prior, the Gaussian distribution can be interpreted as imposing L2 regularization, while the Laplace distribution corresponds to L1 regularization.

The likelihood for regression problems is defined as $\mathbb{P}(y_i|x_i, \omega) = \mathcal{N}(y_i; f^{\omega}(x_i), \beta^{-1})$. The goal is again to compute the posterior distribution $\mathbb{P}(\omega|X, Y)$ by the following relation:

$$\mathbb{P}(\omega|X, Y) \propto \mathbb{P}(\omega) \prod_{i=1}^{N} \mathbb{P}(y_i|x_i, \omega)$$

Bayesian Neural Networks (BNNs) utilize Gaussian distributions for the prior and likelihood but result in a non-Gaussian posterior distribution due to the non-linear dependencies in the neural network. This non-linear dependency is represented by the function $f^{\omega}(x)$, which maps weights to predictions.

Since the true predictive distribution is represented by this integral:

$$\mathbb{P}(y^*|x^*, D) = \int \mathbb{P}(y^*|x^*, \omega)\mathbb{P}(\omega|D)d\omega$$

One can think of Monte Carlo estimation of this integral:

$$\mathbb{P}(y^*|x^*, D) \approx \frac{1}{S} \sum_{s=1}^{S} \mathbb{P}(y^*|x^*, \omega^s) \quad \text{where} \quad \omega^s \sim \mathbb{P}(\omega|D)$$

Or if it is not possible to sample directly from $\mathbb{P}(\omega|D)$, methods such as Monte Carlo Markov Chains (MCMC) Robert and Casella [1999], Metropolis-Hasting (MH) Metropolis et al. [1953], Hastings [1970], and Hamiltonian Monte Carlo (HMC) Duane et al. [1987], Neal [2011] can be used for approximation. However, these methods are effective for accurate posterior inference but have the drawback of not scaling well with large datasets.

Variational Inference

Variational Inference (VI) in Bayesian Neural Networks (BNNs) offers a practical approach to approximate the posterior distribution, as discussed in various influential works Graves [2011], Margossian et al. [2024], Jordan et al. [1999]. The core idea of VI involves approximating the true posterior distribution of the network weights with an *approximating variational distribution*, denoted as $q_\theta(\omega)$ and parameterized by θ, which are referred to as the *variational parameters.*

A critical aspect of VI is the choice of the variational family for $q_\theta(\omega)$. A common approach is to use a Gaussian distribution for each weight, where the mean and variance serve as the variational parameters to be optimized. This choice strikes a balance between flexibility and computational tractability. Thus, the variational approximate posterior $q_\theta(\omega)$ is typically defined as a fully factorized Gaussian:

$$q_\theta(\omega) = \mathcal{N}(\omega|\theta) = \mathcal{N}(\omega|\mu, \Sigma) = \prod_{i=1}^{D} \mathcal{N}(\omega_i|\mu_i, \sigma_i)$$

Here, the variational parameters θ consist of the set $\{(\mu_j, \sigma_j)\}_{j=1}^{D}$, where μ_j and σ_j represent the mean and variance for each weight ω_j in the network.

The objective of VI is to minimize the Kullback-Leibler (KL) divergence between the approximate posterior $q_\theta(\omega)$ and the true posterior distribution:

$$KL(q_\theta(\omega) \parallel \mathbb{P}(\omega|X, Y)) = \int q_\theta(\omega) \log \frac{q_\theta(\omega)}{\mathbb{P}(\omega|X, Y)} \, d\omega \tag{3.1}$$

We can compare the solution for Logistic Regression (LR) against Bayesian Logistic Regression (BLR) to illustrate the differences in the approximations and the impact on classification boundaries:

The LR model outputs a single, fixed decision boundary, assuming certainty in its parameter estimates and therefore not providing insight into epistemic uncertainty. In contrast, the

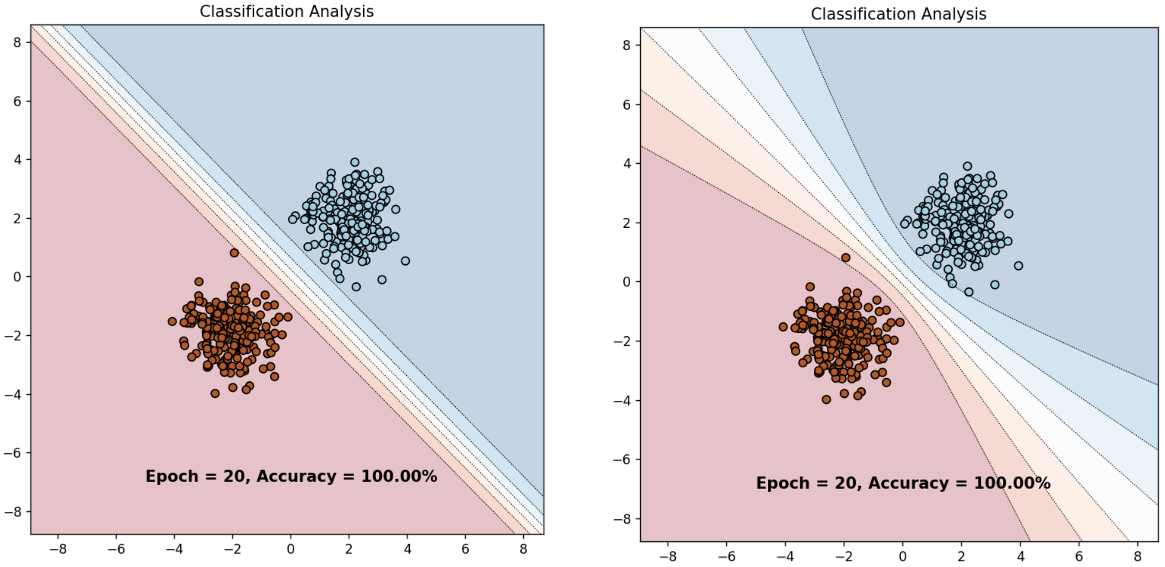


Figure 3.2: Comparison of classification boundaries obtained by MAP solutions for LR and BLR. The left image shows the result for LR, and the right image for BLR

BLR model represents a distribution of possible decision boundaries, allowing it to quantify epistemic uncertainty by considering the variability in the model parameters. And, we recover once again the idea that, the farther away we are from the training data, the higher the epistemic uncertainty, and it should vanish as the number of data goes to infinity providing a more confident decision boundary.

Minimizing the KL divergence in 3.1 is one approach to approximate posterior distributions Depeweg et al. [2017]. This can be extended to the more general α-divergence Póczos and Schneider [2011], Minka [2005]:

$$D_\alpha[\mathbb{P}(\omega|X,Y) \parallel q_\theta(\omega)] = \frac{1}{\alpha(\alpha-1)}\left(1 - \int \mathbb{P}(\omega|X,Y)^\alpha q_\theta(\omega)^{(1-\alpha)}\, d\omega\right),$$

where α is a parameter to be chosen.

Remark:

- $\alpha = 1$ corresponds to the KL divergence.
- $\alpha = 0$ corresponds to the reverse KL divergence.
- $\alpha = 0.5$ corresponds to the squared Hellinger distance.

Beyond epistemic uncertainty captured by Variational Inference (VI), Bayesian Neural Networks (BNNs) can also account for aleatoric uncertainty by introducing latent variables (LVs) Depeweg et al. [2017].

Given a dataset $D_{tr} = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^{N}$, where $\mathbf{x}_n \in \mathbb{R}^D$ and $\mathbf{y}_n \in \mathbb{R}^K$, we model the targets as:

$$\mathbf{y}_n = f(\mathbf{x}_n, \mathbf{z}_n; \omega) + \epsilon_n,$$

where $f(\cdot, \cdot; \omega)$ represents the output of a neural network with weights ω. Here, $\mathbf{z}_n \sim \mathcal{N}(0, \gamma)$ are latent variables, and $\epsilon_n \sim \mathcal{N}(0, \Sigma)$ is additive noise with diagonal covariance matrix Σ.

The latent variables $\mathbf{z}_n$ capture unobserved stochastic features, while ω accounts for model parameters. The posterior distribution $\mathbb{P}(\omega, \mathbf{z}|D_{tr})$ is approximated by:

$$q(\omega, \mathbf{z}) = q(\omega) \times q(\mathbf{z}),$$

where

$$q(\omega) = \prod_{l=1}^{L} \prod_{i=1}^{V_l} \prod_{j=1}^{V_{l-1}+1} \mathcal{N}(\omega_{ij,l} | m_{ij,l}^{\omega}, v_{ij,l}^{\omega}),$$

$$q(\mathbf{z}) = \prod_{n=1}^{N} \mathcal{N}(\mathbf{z}_n | m_n^{\mathbf{z}}, v_n^{\mathbf{z}}).$$

These parameters are optimized by minimizing the divergence between $\mathbb{P}(\omega, \mathbf{z}|D_{tr})$ and the approximation $q(\omega, \mathbf{z})$.

For a new data point x_*, the predictive distribution is given by:

$$p(y_*|x_*, D_{tr}) = \int \int p(y_*|\omega, x_*, z_*) p(z_*) dz_* \, p(\omega, z|D_{tr}) d\omega dz.$$

Here, the posterior distribution is:

$$p(\omega, z|D_{tr}) = \frac{p(y_*|\omega, x_*, z_*) p(z_*) p(\omega)}{p(Y|X)}.$$

In practice, q can be tuned using black-box α-divergence minimization Hernández-Lobato et al. [2016]. The BNN+LV framework effectively captures complex stochastic patterns while managing model uncertainty. This is achieved by jointly learning $q(\mathbf{z})$, which represents the latent variables, and $q(\omega)$, which encodes the uncertainty in model parameters. This approach enables the learning of conditional distributions with complex stochasticity, such as bimodal or heteroscedastic noise.

Following the equations 2.3 and 2.4, the uncertainties can then be disentangled as follows:

$$\mathbb{H}[y^*|x^*, D_{tr}] - \mathbb{E}_{q(\omega)}[\mathbb{H}(y^*|\omega, x^*)] = I(y^*, \omega|x^*, D_{tr}),$$

$$\sigma^2(y^*|x^*, D_{tr}) = \sigma^2_{q(\omega)}(\mathbb{E}[y^*|\omega, x^*]) + \mathbb{E}_{q(\omega)}[\sigma^2(y^*|\omega, x^*)].$$

3.1.2 Sampling Method: Monte Carlo Dropout

Monte Carlo Dropout, introduced by Gal and Ghahramani [2016], is a technique that estimates uncertainty in neural network predictions by applying dropout during both training and inference. Dropout, which randomly omits a subset of neurons during training, simulates various network architectures and helps prevent overfitting by reducing the co-adaptation of neurons, thus improving model generalization. Specifically, a proportion of the elements in the input vector is randomly set to zero, effectively sampling from a Bernoulli distribution with a dropout probability p, typically set to $p = 0.3 - 0.5$.

During training, the neural network is optimized with dropout applied to the weights ω. At inference time, instead of disabling dropout, it remains active, and predictions are made by averaging the outputs from multiple forward passes, each using a different randomly sampled dropout mask.

The predictive distribution for a new data point * is then approximated using Monte Carlo sampling as follows:

$$\mathbb{P}(y^*|x^*, D_{tr}) \approx \frac{1}{S}\sum_{s=1}^{S} f^{\hat{\omega}^{(s)}}(x^*),$$

where $\hat{\omega}^{(s)}$ are weights sampled from the Bernoulli distribution $q(\omega)$ during each forward pass s.

Monte Carlo Dropout provides a practical approximation of Bayesian inference. By averaging predictions over multiple stochastic forward passes, the model captures the uncertainty inherent in its predictions, akin to sampling from the posterior distribution of the weights. This approach has been shown to be effective in various applications, providing a more robust estimation of the model's predictions and their associated uncertainties Gal [2016].

Now, to capture both epistemic and aleatoric uncertainties into a single framework we can consider the methodology from Kendall and Gal [2017]. In 2017, the classical approaches to Bayesian deep learning typically captured either epistemic uncertainty or aleatoric uncertainty alone, depending on where the prior distribution is placed. Placing the prior over the model's weights captures epistemic uncertainty by indicating the variation in weights given the data. Conversely, placing the prior over the model's output captures aleatoric uncertainty.

For regression tasks, the likelihood is often modeled as a Gaussian with mean given by the model output:

$$\mathbb{P}(y|f^{\omega}(x)) = \mathcal{N}(f^{\omega}(x), \sigma^2),$$

where σ represents observation noise. The negative log-likelihood is:

$$-\log \mathbb{P}(y_i|f_{\widetilde{\omega}_i}(x_i)) \propto \frac{1}{2\sigma^2}\|y_i - f_{\widetilde{\omega}_i}(x_i)\|^2 + \frac{1}{2}\log \sigma^2.$$

The predictive variance captures both aleatoric and epistemic uncertainties:

$$\mathrm{Var}(y) \approx \underbrace{\sigma^2}_{\text{Aleatoric}} + \underbrace{\frac{1}{T}\sum_{t=1}^{T} f_{\hat{\omega}_t}(x)^T f_{\hat{\omega}_t}(x) - \left(\frac{1}{T}\sum_{t=1}^{T} f_{\hat{\omega}_t}(x)\right)^2}_{\text{Epistemic}}.$$

Furthermore, in heteroscedastic regression, we consider observation noise $\sigma(x)$ that varies with input x. The log-likelihood is:

$$\log \mathbb{P}(y|f(x)) = \log\left(\frac{1}{\sqrt{2\pi\sigma(x)}}\right) - \frac{\|y - f(x)\|^2}{2\sigma(x)^2},$$

where $\sigma(x)$ is learned as a function of the data. The corresponding loss function is:

$$\mathcal{L}_{\mathrm{NN}}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{2\sigma(x_i)^2}\|y_i - f(x_i)\|^2 + \frac{1}{2}\log \sigma(x_i)^2\right). \tag{3.2}$$

This incorporates *loss attenuation*, which prevents data uncertainty from growing uncontrollably, making the model more robust to noisy or uncertain inputs.

In Bayesian models, we draw weights from an approximate posterior $\hat{\omega} \sim q(\omega)$ to obtain both predictive mean and variance:

$$[\hat{y}, \hat{\sigma}^2] = f^{\hat{\omega}}(x),$$

with the loss function:

$$\mathcal{L}_{BNN}(\theta) = \frac{1}{D}\sum_{i}\left(\frac{1}{2}\hat{\sigma}_i^{-2}\|y_i - \hat{y}_i\|^2 + \frac{1}{2}\log \hat{\sigma}_i^2\right),$$

where $\hat{\sigma}_i^2$ captures the predicted variance, ensuring robustness against noisy data.

The uncertainty in the prediction is given by:

$$\mathrm{Var}(y) \approx \underbrace{\frac{1}{T}\sum_{t=1}^{T}\hat{y}_t^2 - \left(\frac{1}{T}\sum_{t=1}^{T}\hat{y}_t\right)^2}_{\text{Epistemic}} + \underbrace{\frac{1}{T}\sum_{t=1}^{T}\hat{\sigma}_t^2}_{\text{Aleatoric}}.$$

A similar approach applies to classification tasks, where model outputs are passed through a softmax function. Aleatoric uncertainty is modeled by fitting a Gaussian distribution to the logits, and Monte Carlo integration over sampled model weights is used to capture epistemic uncertainties in the predictions.

Bayesian methods offer a principled framework for uncertainty estimation, but applying them to large-scale neural networks presents challenges due to computational complexity. Techniques like posterior tempering [Wenzel et al., 2020] and prior downweighting [Zhang et al., 2018] make Bayesian methods feasible but deviate from pure Bayesian theory. Additionally, defining appropriate priors for complex models is challenging, and model misspecification can lead to suboptimal performance. Despite their theoretical appeal, practical implementation of Bayesian methods in deep learning often requires compromises [Wenzel et al., 2020, Masegosa, 2020].

3.2 Ensembles

The Bayesian approach of integrating over a distribution of parameters for predictions is conceptually similar to ensembling, where multiple models with different parameter settings are aggregated to improve robustness and capture uncertainty [Dietterich, 2000]. For example, Monte Carlo Dropout [Gal and Ghahramani, 2016] approximates Bayesian inference by applying dropout during both training and inference. Ensembling extends this idea by combining predictions from several independently trained models with different parameter settings $\{f^{\omega^{(i)}}\}_{i=1}^{M}$, and the final prediction is typically the average of these models:

$$\hat{y} = \frac{1}{M}\sum_{i=1}^{M} f^{\omega^{(i)}}(\mathbf{x}),$$

providing a more robust estimate of the predictive distribution. Bagging (Bootstrap Aggregating) [Breiman, 1996], a key ensemble method, involves training multiple models on different bootstrap samples of the data and averaging their predictions to reduce variance. Each model is trained on a different sample, and the predictions are averaged to form the final output. Deep Ensembles [Lakshminarayanan et al., 2017], on the other hand, involve training multiple neural networks independently, each starting from different random initializations. This approach captures uncertainty effectively, as each model learns different aspects of the data. Inspired by Kendall and Gal [2017], each model can output both a mean prediction $\mu^{\omega^{(i)}}(\mathbf{x})$ and a variance $\sigma^{2,\omega^{(i)}}(\mathbf{x})$, representing a Gaussian distribution over the outputs. The total predictive uncertainty is also computed using the law of total variance 2.4:

$$\mathrm{Var}(\hat{y}) = \underbrace{\frac{1}{M}\sum_{i=1}^{M}\sigma^{2,\omega^{(i)}}(\mathbf{x})}_{\text{Aleatoric}} + \underbrace{\frac{1}{M}\sum_{i=1}^{M}\mu^{\omega^{(i)}}(\mathbf{x})^2 - \left(\frac{1}{M}\sum_{i=1}^{M}\mu^{\omega^{(i)}}(\mathbf{x})\right)^2}_{\text{Epistemic}}.$$

For classification tasks, the predictive entropy $\mathbb{H}(\hat{y})$ can be computed as:

$$\mathbb{H}(\hat{y}) = -\sum_{c=1}^{C}\hat{p}_c \log \hat{p}_c,$$

where $\hat{p}_c = \frac{1}{M}\sum_{i=1}^{M}\mathrm{Softmax}(f^{\omega^{(i)}}(\mathbf{x}))_c$, the average predicted probability for class c across the ensemble. This entropy serves as a measure of uncertainty, with higher values indicating greater uncertainty in predictions. The total predictive entropy $\mathbb{H}(\hat{y})$ can be decomposed to disentangle aleatoric and epistemic uncertainties 2.3:

$$\mathbb{H}(\hat{y}) = \underbrace{\mathbb{E}[\mathbb{H}(\hat{y}|\omega)]}_{\text{Aleatoric Uncertainty}} + \underbrace{\mathbb{I}(\hat{y};\omega)}_{\text{Epistemic Uncertainty}}.$$

Other ensemble techniques, such as SGD Ensembles [Wilson and Izmailov, 2020], which exploit stochasticity in training, and Snapshot Ensembles [Huang et al., 2017], which save model weights at different stages of training, also contribute to uncertainty quantification but are not explored in detail here.

Deep Ensembles consistently outperform other uncertainty estimation methods, particularly under dataset shift, thanks to their model diversity. Each ensemble member, initialized differently, explores unique regions of the loss landscape, enhancing both robustness and calibration. Valdenegro-Toro and Saromo Mori [2022] demonstrate that Deep Ensembles excel at disentangling aleatoric and epistemic uncertainties and provide superior robustness to out-of-distribution (OOD) samples. However, the approach requires training multiple models (typically 5 to 10), leading to substantial computational and memory overhead, which limits scalability for real-time applications. Additionally, inference demands multiple forward passes, making this method computationally expensive for UQ.

3.3 Hierarchical Methods - Evidential Deep Learning

Evidential Deep Learning (EDL) offers a single-model approach for uncertainty estimation, avoiding the computational cost of training multiple models and generating diverse predictions. EDL enables uncertainty estimation within a single forward pass by parameterizing distributions over distributions, applicable in both classification and regression.

Returning to the probabilistic modeling of uncertainty 3.3:

$$\mathbb{P}(y^*|x^*, X, Y) = \int \underbrace{\mathbb{P}(y^*|x^*, \omega)}_{\text{Aleatoric}} \underbrace{\mathbb{P}(\omega|X, Y)}_{\text{Epistemic}} \, d\omega.$$

Traditional predictive uncertainty estimation often requires multiple parameter sets and can only approximate the predictive posterior. However, it is possible to further factorize this equation to obtain a tractable form, as demonstrated by Malinin and Gales [2018]:

$$\mathbb{P}(y|\mathbf{x}, \mathcal{D}_{tr}) = \int\int \underbrace{\mathbb{P}(y|\pi)}_{\text{Aleatoric}} \underbrace{\mathbb{P}(\pi|\mathbf{x}, \omega)}_{\text{Distributional}} \underbrace{\mathbb{P}(\omega|\mathcal{D}_{tr})}_{\text{Epistemic}} \, d\pi \, d\omega \approx \int \underbrace{\mathbb{P}(y|\pi)}_{\text{Aleatoric}} \underbrace{\mathbb{P}(\pi|\mathbf{x}, \hat{\omega})}_{\text{Distributional}} \, d\pi,$$

with $\mathbb{P}(\omega|\mathcal{D}_{tr}) \approx \delta(\omega - \hat{\omega})$.

In this context, distributional uncertainty arises from the mismatch between training and test distributions, as discussed by Malinin and Gales [2018]. By using a point estimate $\hat{\omega}$ for the intractable integral, the problem reduces to a solvable form for the Dirichlet distribution, distinguishing between ambiguous data and data from different distributions. EDL aims to model a distribution over distributions. In classification, a neural network typically outputs a softmax vector, a discrete probability distribution over classes. EDL parameterizes a second-order distribution over this simplex, outputting the parameters of a Dirichlet distribution rather than softmax probabilities, thereby representing second-order uncertainties.

This concept stems from the notion of a conjugate prior, leading to tractable posterior computation. For instance, a binary classification problem is often modeled using the Bernoulli likelihood:

$$\mathbb{P}(y|\pi) = \pi^y (1 - \pi)^{(1-y)}.$$

In Bayesian inference, the Beta distribution is a common prior for a Bernoulli likelihood, characterized by two shape parameters α_1 and α_2:

$$\mathbb{P}(\pi; \alpha_1, \alpha_2) = \frac{1}{B(\alpha_1, \alpha_2)} \pi^{\alpha_1 - 1} (1 - \pi)^{\alpha_2 - 1},$$

where $B(\alpha_1, \alpha_2) = \frac{\Gamma(\alpha_1)\Gamma(\alpha_2)}{\Gamma(\alpha_1 + \alpha_2)}$. When extending to multiple classes, a Categorical likelihood is used:

$$\mathbb{P}(y|\pi) = \prod_{k=1}^{K} \pi_k^{\mathbb{1}[y=k]},$$

where π is a vector of class probabilities. The Dirichlet distribution then serves as a suitable prior, generalizing the Beta distribution to multiple categories:

$$\mathbb{P}(\pi; \alpha) = \frac{1}{B(\alpha)} \prod_{k=1}^{K} \pi_k^{\alpha_k - 1}$$

For classification with K classes, a neural classifier is usually a function $f_\omega : \mathbb{R}^D \to \mathbb{R}^K$, mapping an input $\mathbf{x}$ to logits, followed by a softmax function to produce a Categorical distribution over classes. The same architecture can be adapted to parameterize a Dirichlet distribution.

In predicting a distribution over Categorical distributions, the Dirichlet network creates a Categorical distribution from the predicted concentration parameters as:

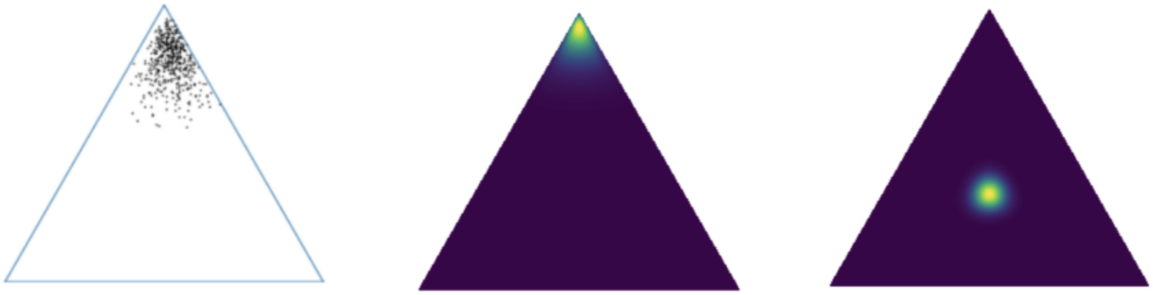
$$\alpha = \exp\left(f_\omega(\mathbf{x})\right), \quad \pi_k = \frac{\alpha_k}{\alpha_0}, \quad \hat{y} = \arg\max_{k \in \mathcal{K}} \pi_k.$$

Data uncertainty, or aleatoric uncertainty, can be quantified by evaluating the expected entropy of the data distribution $\mathbb{P}(y|\pi)$:

$$\mathbb{E}_{\mathbb{P}(\pi|\mathbf{x}, \hat{\omega})} \left[\mathbb{H} \left(\mathbb{P}(y|\pi) \right) \right] = - \sum_{k=1}^{K} \frac{\alpha_k}{\alpha_0} \left(\psi(\alpha_k + 1) - \psi(\alpha_0 + 1) \right),$$

where $\psi(x)$ is the digamma function.

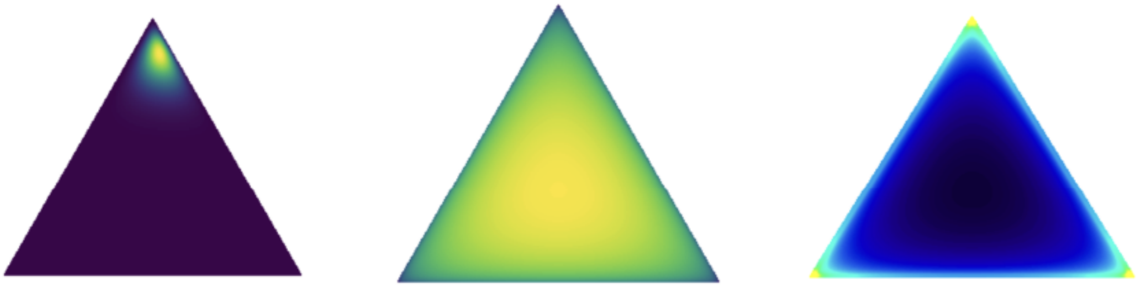
Distributional uncertainty is another key property, distinguishing uncertainty due to model underspecification from uncertainty due to unknown inputs. This can be expressed through the mutual information as introduced previously 2.3 between the label y and its Categorical



(a) Categorical distributions predicted by a neural ensemble on the probability simplex.

(b) Probability simplex for a confident prediction, for with the density concentrated in a single corner.

(c) Dirichlet distribution for a case of data uncertainty, with the density concentrated in the center.



(d) Dirichlet distribution for a case of model uncertainty, with the density spread out more.

(e) Dirichlet for a case of distributional uncertainty, with the density spread across the whole simplex.

(f) Alternative approach to distributional uncertainty called representation gap, with density concentrated along the edges.

Figure 3.3: Image source: Ulmer et al. [2021]

distribution π:

$$I\left[y, \pi \mid \mathbf{x}, \mathcal{D}\right] = \mathbb{H}\left[\mathbb{E}_{\mathbb{P}(\pi \mid \mathbf{x}, \mathcal{D})}\left[\mathbb{P}(y \mid \pi)\right]\right] - \mathbb{E}_{\mathbb{P}(\pi \mid \mathbf{x}, \mathcal{D})}\left[\mathbb{H}\left[\mathbb{P}(y \mid \pi)\right]\right]. \tag{3.3}$$

This mutual information quantifies how much information about π is gained by knowing y. In well-defined regions, this information should be low, indicating low distributional uncertainty. However, in regions with sparse data, this mutual information indicates higher uncertainty.

Extending EDL to regression problems is feasible, though the Dirichlet distribution is not suitable. Regression problems typically use a normal likelihood, and several prior distributions are applicable. For example, Amini et al. [2020] and Charpentier et al. [2022] use the Normal-Inverse Gamma distribution, leading to a scaled inverse-χ^2 posterior, while others opt for a Normal-Wishart prior. In univariate regression, the problem is modeled as a normal distribution with unknown mean and variance:

$$\mathbb{P}(y; \pi, \sigma^2) = \mathcal{N}(y; \pi, \sigma^2),$$

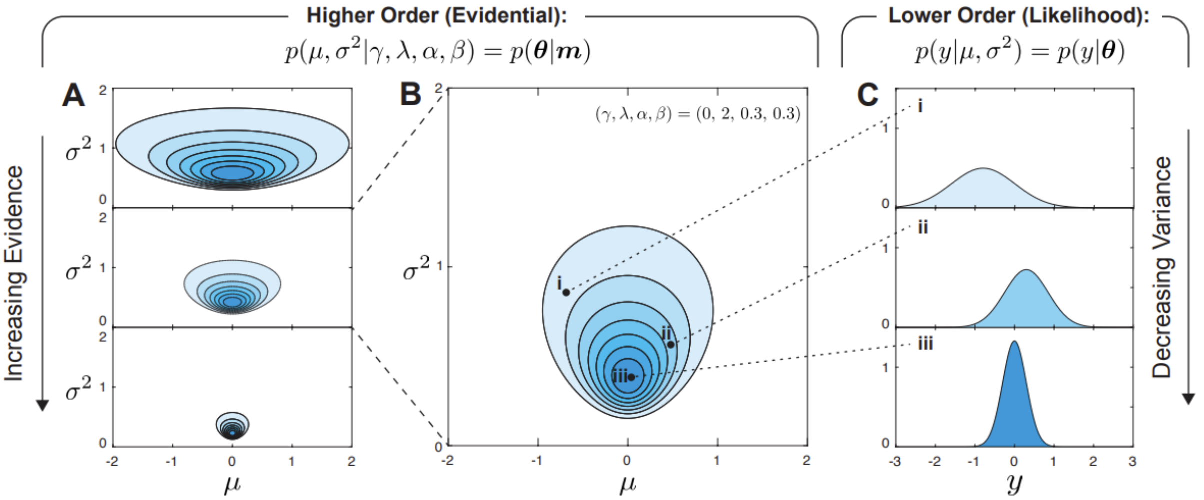


Figure 3.4: (Image source: Amini et al. [2020]).

with a normal prior for the mean and an inverse Gamma prior for the variance:

$$\mathbb{E}[\sigma^2] = \frac{\beta}{\alpha - 1}, \quad \mathrm{Var}[\pi] = \frac{\beta}{\nu(\alpha - 1)}.$$

These predictions are made by different "heads" of a neural network, allowing for the estimation of both aleatoric and epistemic uncertainties using closed-form solutions.

However, concerns have been raised about Evidential Deep Learning (EDL) and second-order distributions. Kopetzki et al. [2021] show that Dirichlet-based uncertainty models, which predict Dirichlet distribution parameters to provide uncertainty estimates, struggle under adversarial attacks. These models have difficulty in correctly identifying misclassified samples, detecting adversarial examples, and distinguishing between in-distribution (ID) and out-of-distribution (OOD) data. Bengs et al. [2023] add that second-order learners, which aim to represent epistemic uncertainty through distributions on probability distributions, face fundamental theoretical shortcomings. They demonstrate that no loss function provides a true incentive for second-order learners to faithfully represent epistemic uncertainty, similar to how proper scoring rules work for first-order learners.

3.4 Other methods

Recent work on uncertainty estimation in neural networks spans several efficient methods. Approaches using Gaussian processes (GPs), such as BNNs converging to GPs [Neal, 1994], underlie methods like SNGP [Liu et al., 2020a] and DUQ [Van Amersfoort et al., 2020], leveraging scalable uncertainty estimation with Gaussian processes. Additionally, Maddox et al. [2019] group SWAG with Gaussian-based approaches, approximating posteriors around SGD iterates. Variational methods for uncertainty, like Variational Bayesian Last Layers (VBLL) [Harrison et al., 2024], focus on Bayesian techniques applied to the last layer, enhancing uncertainty estimates with minimal computational cost. Techniques like BatchEnsemble [Wen et al., 2020] further improve scalability by sharing weight matrices across ensemble members. Distance-based uncertainty estimators, such as Mahalanobis distance-based detection [Lee et al., 2018], also offer robust solutions for out-of-distribution detection and adversarial robustness, achieving state-of-the-art performance with low computational overhead. Another technique, AugMix [Hendrycks et al., 2020], focuses on improving model robustness and uncertainty estimates by employing stochastic data augmentations and mixing strategies, resulting in better calibrated models under severe distributional shifts.

To address the scalability issues typically faced by ensemble methods, Havasi et al. [2020] and Tran et al. [2020] propose two efficient alternatives. The Multi-Input Multi-Output (MIMO) model processes multiple inputs in a single forward pass and produces multiple outputs by learning implicit subnetwork paths, thus avoiding the need for explicit ensembles or low-rank perturbations. This approach significantly improves computational efficiency while leveraging over-parameterized architectures like WideResNet and ResNet. Hydra, on the other hand, distills ensembles into a single multi-headed network, maintaining the diversity of ensemble predictions. Both methods capture uncertainty in in-domain and out-of-distribution tasks while reducing computational costs compared to traditional Deep Ensembles. However, Hydra still requires training both the ensemble and the distilled model, while MIMO necessitates training a new model, meaning they cannot be applied directly to pre-trained models without modifying their architecture.

To avoid these limitations, Mi et al. [2022] introduce a training-free, model-agnostic uncertainty estimation method for dense regression, which can estimate uncertainty on a pre-trained model without retraining or redesign. However, these methods still require multiple forward passes—whether through input perturbations or weight sampling—to generate uncertainty estimates effectively.

This raises the question: can uncertainty be assessed in a model-agnostic, post-hoc and single-pass setting?

3.5 Post-hoc Single-Pass Uncertainty Quantification methods

Conformal prediction provides a framework for generating prediction intervals with guaranteed coverage, offering reliable uncertainty estimates by calibrating model outputs based on prior data [Shafer and Vovk, 2008]. However, its inability to separate aleatoric from epistemic uncertainties is a significant drawback, as this distinction is essential for understanding the sources of uncertainty—whether they arise from inherent data noise or model limitations [Angelopoulos and Bates, 2021]. Furthermore, conformal prediction assumes that the data distribution remains consistent between training and testing phases, limiting its effectiveness under dataset shifts. It also tends to produce overly conservative estimates, often resulting in wide prediction intervals, particularly in high-dimensional spaces.

In contrast, temperature scaling focuses specifically on recalibrating a model's predicted probabilities post-hoc. This technique, introduced by Guo et al. [2017], adjusts the softmax outputs through a scalar temperature parameter T, optimized on a calibration set to reduce the Expected Calibration Error (ECE). The adjusted probabilities follow:

$$\mathbb{P}(y_i \mid \mathbf{x}) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

where z_i are the logits for each class. Temperature scaling effectively improves the alignment of predicted confidence with actual accuracy, but it does not provide a direct measure of uncertainty or differentiate between types of uncertainty.

Both conformal prediction and temperature scaling share notable limitations. Neither can disentangle epistemic and aleatoric uncertainty, and both assume a consistent data distribution, reducing their robustness under dataset shifts. Additionally, while conformal prediction offers uncertainty estimates, its prediction intervals can be overly wide, and temperature scaling, though improving calibration, may struggle with overlapping data distributions, as highlighted by Chidambaram and Ge [2024].

On the other hand, a method that quantifies uncertainties more effectively in a post-hoc manner is the one of Shen et al. [2022]. The paper addresses the challenge of post-hoc UQ by enhancing the uncertainty quality of a pre-trained model without compromising its predictive performance. More recent meta-modeling approaches Chen et al. [2019], Jain et al. [2021] attempt to predict the correctness of the pre-trained model's predictions, yet they rely on point estimates of the meta-model parameters, which can be unreliable when the validation set is small.

To overcome these limitations, the authors propose a Bayesian meta-model approach. By employing a Dirichlet meta-model, the method should capture both total and epistemic uncertainty. The proposed meta-model employs several linear layers, each connected to different intermediate layers of the base model, with a final linear layer that consolidates all features into a single output. Specifically, given an input x, the intermediate feature representations extracted from the base model are denoted as $\{\Phi_j(x)\}_{j=1}^m$. Each intermediate feature Φ_j is passed through a linear layer that generates a low-dimensional meta-feature $\{g_j(\Phi_j(x))\}_{j=1}^m$. The final linear layer of the meta-model then takes these meta-features as input to produce a single output, expressed as $\tilde{y} = g_c(\{g_j(\Phi_j(x))\}_{j=1}^m; \omega)$, where g_c is the final linear layer with weights ω. These linear layers are composed of fully connected layers and activation functions, resulting in a simpler structure that facilitates efficient training.

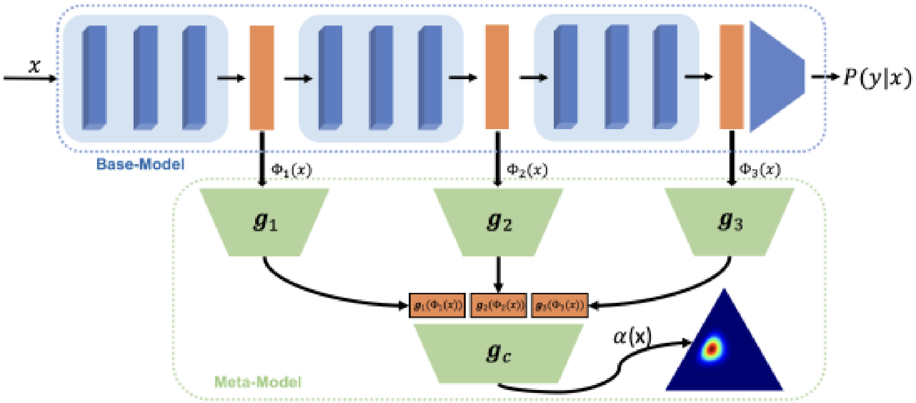


Figure 3.5: Meta-Model structure (Image source: Shen et al. [2022]).

When an input x is fed into the base model, it outputs a conditional label distribution $\mathbb{P}_B(y|\Phi(x)) \in \Delta^{K-1}$, which corresponds to a point on the probability simplex. However, this distribution represents only the model's uncertainty regarding class labels and fails to capture epistemic uncertainty, which arises from insufficient knowledge about a sample. To better quantify epistemic uncertainty, the authors adopt a Dirichlet-based approach close to EDL with a similar training methodology of the parameters of the Dirichlet distribution and a quantification of uncertainties as mentioned in 3.3 with slight modifications regarding the different features collected as inputs:

$$I\left(y, \pi \mid \Phi(\mathbf{x}_i)\right) = \underbrace{\mathcal{H}\left(\mathbb{E}_{\mathbb{P}(\pi \mid \Phi(\mathbf{x}); \omega_g)}\left[\mathbb{P}(y \mid \pi)\right]\right)}_{\text{Total Uncertainty}} - \underbrace{\mathbb{E}_{\mathbb{P}(\pi \mid \Phi(\mathbf{x}); \omega_g)}\left[\mathcal{H}\left(\mathbb{P}(y \mid \pi)\right)\right]}_{\text{Aleatoric Uncertainty}} \tag{3.4}$$

However, as mentioned previously, second-order distributions face challenges, as shown by Bengs et al. [2023], highlighting fundamental issues, noting that second-order learners lack proper incentives to accurately represent epistemic uncertainty. Another issue to highlight is that this method needs a careful adaptation of the meta-model to the specific pretrained model.

Chapter 4

Proposed Method

To address the research gap, we adopt a post-hoc approach. Using a pretrained neural network named the *base model*, our goal is to train a meta-model on top of it, enabling uncertainty quantification (UQ) in a single forward pass. Ahn et al. [2023] demonstrated that the number of activated neurons in the penultimate layer differs significantly between in-distribution (ID) and out-of-distribution (OOD) data. Specifically, most neurons are activated by ID samples (e.g., ImageNet), whereas fewer neurons are triggered when processing OOD datasets like Textures and iNaturalist. An activated neuron is defined as one with an activation value greater than zero. This insight is crucial for our research, as it introduces a novel methodology for OOD detection by analyzing neuron activations in deep neural networks. Based on this, we train our meta-model on the activations of the penultimate layer.

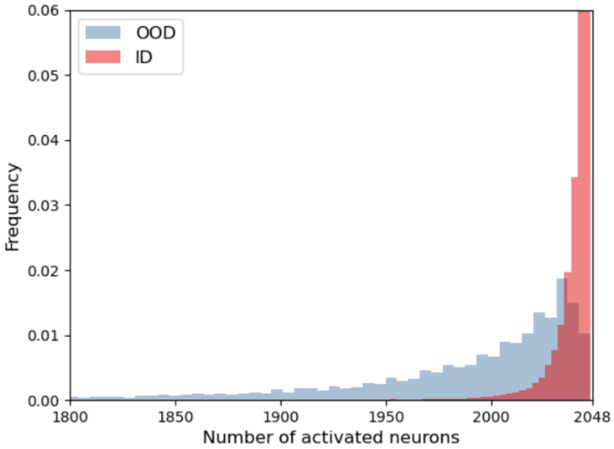


Figure 4.1: Number of activated neurons in the penultimate layer (Image source: Ahn et al. [2023]).

The hypothesis is that for ID data, the meta-model will recognize patterns with a higher number of active neurons, while for OOD data, with fewer activated neurons, it will generate anomalous outputs, indicating uncertainty. To leverage this knowledge effectively, we use ReLU activation functions, which zero out inactive neurons, amplifying the distinction between ID and OOD data by accentuating the difference in the number of activated neurons.

Osband et al. [2023] highlight that an important factor in disentangling epistemic and aleatoric uncertainties is the joint distribution. Learning the joint distribution during training allows for more precise handling of uncertainty in predictions. As noted by Havasi et al. [2020], training multiple subnetworks with a summed loss is equivalent to training a single large network that learns a joint distribution. Based on this insight, we designed an architecture called the *Multi-output module*, which builds an ensemble of last layers with a shared architecture part. These layers act on the activations of the penultimate layer, processing multiple inputs and generating multiple outputs in a single forward pass.

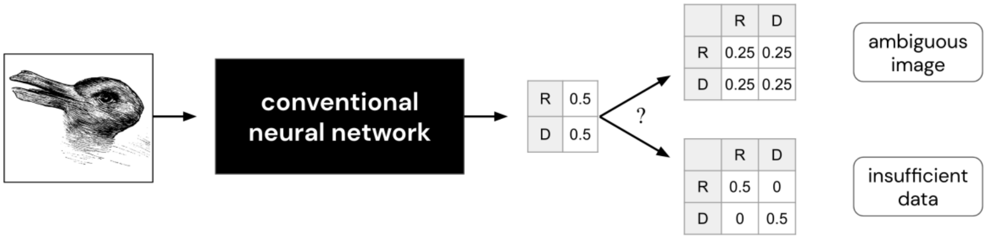


Figure 4.2: Conventional neural networks generate marginal predictions that do not separate true ambiguity from insufficient data. Joint predictions can make this distinction clearer. (Image source: Osband et al. [2023]).

4.1 Method Description

We focus on classification problems in this paper due to time constraints, although a similar approach could be applied to regression tasks. Let $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ denotes the input space, and $\mathcal{Y} = \{1, \dots, K\}$ represents the label space. Given a base-model training set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N \in \mathcal{Z}^N$, which consists of i.i.d. samples. A pretrained base model $h_{ll} \circ \varphi : \mathcal{X} \to \Delta^{K-1}$ is already trained on this training set. Here, $\varphi : \mathcal{X} \to \mathbb{R}^l$ represents the penultimate layer feature extractor of the base model, and $h_{ll} : \mathbb{R}^l \to \Delta^{K-1}$ denotes the *last layer* that outputs the predicted label distribution $\mathbb{P}_B(y \mid \varphi(x)) \in \Delta^{K-1}$ given the input x. where Δ^{K-1} denotes the $(K-1)$-dimensional probability simplex, which is the set of all non-negative vectors in $\mathbb{R}^K$ whose entries sum to 1.

In more detail, we first denote θ, θ_{ll} and ω respectively the weights of the base model up to the last layer, the weights of the last layer and the weights of the multi output module. Specifically, the penultimate layer output is given by $\varphi(x) = \varphi_\theta(x) = \varphi(x;\theta)$, and the predicted label distribution is $h(\varphi(x)) = h(\varphi(x;\theta);\theta_{ll})$.

The performance of the base model is evaluated using a non-negative loss function $\ell_B : \mathbb{W} \times \mathcal{Z} \to \mathbb{R}_+$ (e.g., cross-entropy loss). The base model is pretrained by minimizing the empirical risk over the dataset $\mathcal{D}$:

$$L_B(h \circ \varphi, \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \ell_B \left(h \circ \varphi, (x_i, y_i)\right).$$

Although the well-trained base model achieves good predictive accuracy, the output distribution $\mathbb{P}_B(y|\varphi(x))$ is typically unreliable for UQ, as it may be overconfident or poorly calibrated. To address this without retraining the base model, we aim to improve UQ in a post-hoc manner by training a meta-model $g : \mathbb{R}^l \to \mathcal{Y}$ with parameters ω, building on top of the base model.

The proposed Multi-output module method is trained on the penultimate layer feature representation $\varphi(x)$. We remove the last layer of the base model and duplicate it M times to create an ensemble of last layers, denoted as $h_i^{input}(\varphi(x))$ for $i = 1, \ldots, M$. Each of these layers, referred to as an *input head*, is a deep copy of the last layer of the base model with weight reinitiated randomly, it takes $\varphi(x)$ as input. The outputs of this ensemble of heads are concatenated as follows:

$$h_{\text{cat}}^{\text{input}}(\varphi(x)) = \left(h_1^{\text{input}}(\varphi(x)), \ldots, h_M^{\text{input}}(\varphi(x))\right) \in \mathbb{R}^{K \times M},$$

which is then processed through a fully connected layer, called the *shared layer*, from $\mathbb{R}^{K \times M}$ to $\mathbb{R}^{K \times M}$ that permits to mix the outputs. The final step is to produce logits using M *output heads*, which are fully connected layers mapping the shared architecture's output from $\mathbb{R}^{K \times M}$ to $\mathbb{R}^K$ for each head all called h_i^{output} for $i = 1$ to M. After this operation one can use a softmax function to map each output heads' logits to a probability space from $\mathbb{R}^K$ to Δ^{K-1} We define M input and output heads because the Multi-output module takes M inputs and produces M outputs.

During training, the penultimate layer features map $\varphi(x_1), \ldots, \varphi(x_M)$ are independently obtained as outputs from the base model on the training set, and the module is trained to classify each corresponding input at the same time.

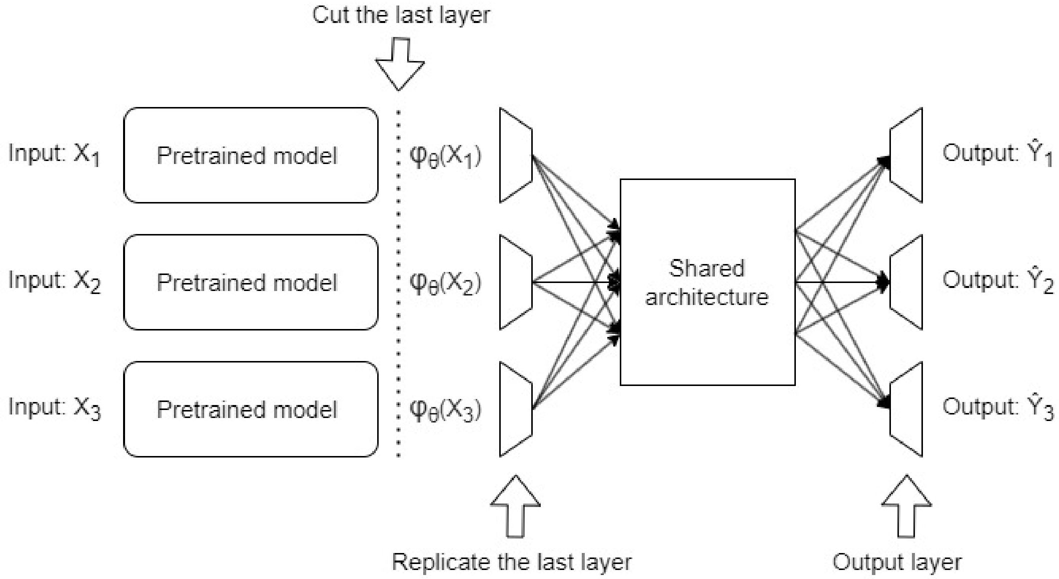


Figure 4.3: Schematic view of the Multi output module training methodology

Our meta-model is learning the link between the "usual" penultimate layer activations patterns of the base model and the corresponding resulting classes, for several inputs and outputs at the same time. This approach is motivated by the need to perform UQ by leveraging a joint distribution while maintaining the capability of using standard metrics with typical model outputs. Following Havasi et al. [2020], the underlying idea is that minimizing a sum of losses corresponds to minimizing the sum of negative log-likelihoods of the predictions plus a regularization term:

$$L_{\text{Module}}(\omega) = \mathbb{E}_{(\mathbf{x}_1,\mathbf{y}_1),\ldots,(\mathbf{x}_M,\mathbf{y}_M)\in\mathcal{D}}\left[\sum_{m=1}^{M} -\log \mathbb{P}_\omega(y_m \mid \varphi(x_1),\ldots,\varphi(x_M))\right] + R(\omega),$$

where the sum of log-likelihoods corresponds to the log-likelihood of the joint distribution:

$$\sum_{m=1}^{M} \log \mathbb{P}_\omega(y_m \mid \varphi(x_1),\ldots,\varphi(x_M)) = \log \mathbb{P}_\omega(y_1,\ldots,y_M \mid \varphi(x_1),\ldots,\varphi(x_M)),$$

since the input-output pairs are independent. The module is trained on M-tuples of independently sampled data points.

At evaluation time, for an unseen input image x' to the base model, the penultimate layer's activations $\varphi(x')$ is given as input to the meta-model and repeated M times, so $\varphi(x_1) = \cdots = \varphi(x_M) = \varphi(x')$. Each output head approximates the predictive distribution $\mathbb{P}_\omega(y_i \mid$

$\varphi(x'), \ldots, \varphi(x'))$, and averaging these predictions yields the final global output:

$$\mathbb{P}_\omega(y' \mid \varphi(x')) = \frac{1}{M}\sum_{i=1}^{M} \mathbb{P}_\omega(y_i \mid \varphi(x'), \ldots, \varphi(x')).$$

We obtain outputs resembling those produced by a Deep Ensemble, and a straightforward measure of uncertainty can be derived by simply calculating the variance of these outputs. In the next section we go further to differentiate the two types of uncertainties given the outputs.

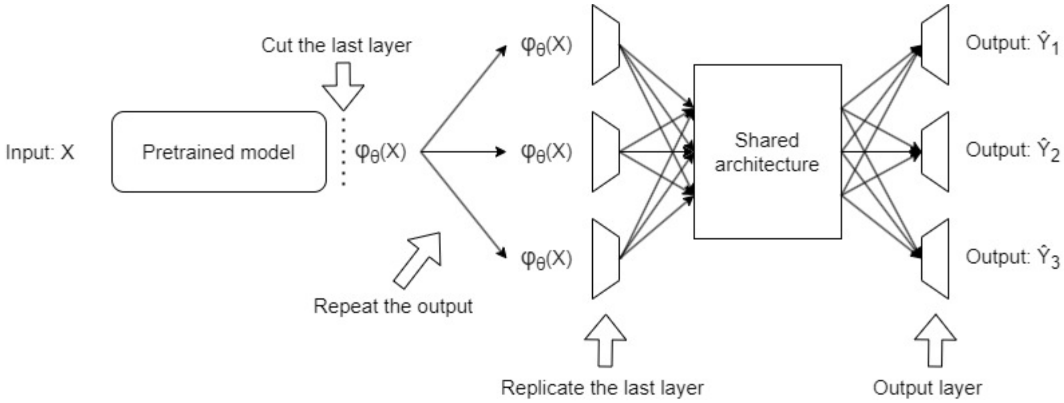


Figure 4.4: Schematic view of the Multi output module inference methodology

You can find a detailed pseudocode of the Multi-output module at the end of the thesis: 1.

4.2 Uncertainty quantification and disentanglement

The key idea of this uncertainty disentanglement is that each of the M softmax outputs corresponds to a distinct head of the model, with each head contributing to the overall uncertainty estimation. We want to decompose uncertainty into aleatoric and epistemic uncertainty.

Aleatoric uncertainty stems from ambiguity in the data, where the model struggles to differentiate between classes due to unclear inputs. This is reflected when the heads provide confident but differing predictions. In contrast, epistemic uncertainty arises from a lack of model knowledge or insufficient training and is captured by high entropy across all heads when they globally agree on their uncertainty.

At inference, we have M softmax outputs from the model heads, represented as:

$$\{\mathbf{p}^m\}_{m=1}^M = \{(p_1^m, p_2^m, \dots, p_K^m)\}_{m=1}^M ,$$

where $\mathbf{p}^m$ is the softmax output from the m-th head, and K is the number of classes.

The mean softmax prediction across all heads is:

$$\bar{\mathbf{p}} = (\bar{p}_1, \dots, \bar{p}_K) = \frac{1}{M}\sum_{m=1}^M \mathbf{p}^m = \left(\frac{1}{M}\sum_{m=1}^M p_1^m, \dots, \frac{1}{M}\sum_{m=1}^M p_K^m\right),$$

where $\bar{p}_i = \frac{1}{M}\sum_{m=1}^M p_i^m$.

The total predictive uncertainty is the entropy of the global prediction, e.g. the mean prediction:

$$\mathbb{H}(\bar{\mathbf{p}}) = -\sum_{i=1}^K \bar{p}_i \log \bar{p}_i.$$

The entropy of the m-th head is given by:

$$\mathbb{H}(\mathbf{p}^m) = -\sum_{i=1}^K p_i^m \log p_i^m.$$

By introducing $\bar{p}_i = \frac{1}{M}\sum_{m=1}^M p_i^m$, we rewrite this as:

$$p_i^m \log p_i^m = p_i^m \log \bar{p}_i + p_i^m \log \frac{p_i^m}{\bar{p}_i},$$

leading to:

$$\mathbb{H}(\mathbf{p}^m) = -\sum_{i=1}^K p_i^m \log \bar{p}_i - \sum_{i=1}^K p_i^m \log \frac{p_i^m}{\bar{p}_i}.$$

Summing over all heads:

$$\frac{1}{M}\sum_{m=1}^M \mathbb{H}(\mathbf{p}^m) = \frac{1}{M}\sum_{m=1}^M \left(-\sum_{i=1}^K p_i^m \log \bar{p}_i - \sum_{i=1}^K p_i^m \log \frac{p_i^m}{\bar{p}_i}\right),$$

Distribute the summation:

$$\frac{1}{M}\sum_{m=1}^M \mathbb{H}(\mathbf{p}^m) = -\sum_{i=1}^K \left(\frac{1}{M}\sum_{m=1}^M p_i^m\right) \log \bar{p}_i - \frac{1}{M}\sum_{m=1}^M \sum_{i=1}^K p_i^m \log \frac{p_i^m}{\bar{p}_i}.$$

It yields:

$$\frac{1}{M}\sum_{m=1}^{M}\mathbb{H}(\mathbf{p}^m) = -\sum_{i=1}^{K}\bar{p}_i\log\bar{p}_i - \frac{1}{M}\sum_{m=1}^{M}\sum_{i=1}^{K}p_i^m\log\frac{p_i^m}{\bar{p}_i}.$$

Thus, we obtain the following decomposition:

$$\mathbb{H}(\bar{\mathbf{p}}) = \underbrace{\frac{1}{M}\sum_{m=1}^{M}\mathbb{H}(\mathbf{p}^m)}_{\text{Epistemic Uncertainty}} + \underbrace{\frac{1}{M}\sum_{m=1}^{M}\mathrm{KL}(\mathbf{p}^m\|\bar{\mathbf{p}})}_{\text{Aleatoric Uncertainty}}.$$

Here, epistemic uncertainty is represented by the average entropy across the heads, while aleatoric uncertainty is quantified by the mean KL divergence between each head's prediction and the average output. If all the outputs confidently agree on a single label (in a one-hot fashion), both uncertainties will be zero. If the heads are close to one-hot but predict different labels, epistemic uncertainty will be low, while aleatoric uncertainty will be high. In the final scenario, where the heads agree on a low-confidence prediction due to insufficient knowledge, epistemic uncertainty will be high and aleatoric uncertainty low.

Chapter 5

Experiments

5.1 Experimental settings

In this section, we present results on image classification tasks using MNIST, CIFAR-10, CIFAR-100 datasets and their corrupted variants Hendrycks and Dietterich [2019]. The corrupted variants of the CIFAR datasets are introduced to evaluate the robustness of machine learning models under various types of noise and distortions that they may encounter in real-world scenarios. These corruptions simulate common challenges such as blur, noise, weather conditions, or digital distortions that can affect image quality.

Specifically, the CIFAR-10 and CIFAR-100 corrupted variants contain 15 different types of corruptions, each applied at five different severity levels. The corruptions are grouped into four categories:

- **Noise**: Random perturbations such as Gaussian noise, shot noise, and impulse noise that distort pixel values.
- **Blur**: Includes Gaussian blur, motion blur, and defocus blur, which reduce the clarity and sharpness of the image.
- **Weather**: Simulates weather conditions like snow, frost, and fog that obscure parts of the image or alter the brightness and contrast.
- **Digital**: Perturbations from digital operations such as JPEG compression, pixelation, and contrast alterations.

These corrupted variants are designed to test how well a model can generalize and maintain performance when exposed to such distortions, making it a critical benchmark for evaluating robustness and UQ. For the CIFAR datasets, we use a Wide ResNet 28-10 as the base model since it achieves over 95% test accuracy on CIFAR-10 with minimal data augmentation Zagoruyko and Komodakis [2016].

We define metrics specific to the corrupted CIFAR datasets below. For general results with these metrics, see the baselines provided in Google's Uncertainty Baselines repository Google Research [2023].

cNLL/cA/cECE: Negative-log-likelihood, accuracy, and calibration error on CIFAR-10-C. The same corruptions are applied to produce CIFAR-100-C, where ‘c‘ stands for corrupted. Results are averaged across corruption intensities and corruption types.

Due to time limitations, we were unable to compare the baselines trained in the exact same setting as our experiments. Instead, for now, we compare our results to the baselines provided by Google's Uncertainty Baselines repository Google Research [2023], which implements MNIST, CIFAR-10 and CIFAR-100 models for uncertainty estimation. While the settings slightly differ, particularly for the base model, the results provide a useful point of reference for evaluating performance across different robustness benchmarks.

5.2 Uncertainty Quantification Experimental results

In the following, we present the results across the three benchmark datasets for varying numbers of heads, averaging the outcomes over 5 runs along with their standard deviations.

5.2.1 Cifar-10

For the Cifar-10 dataset just like in Google's Uncertainty Baselines repository Google Research [2023] we consider the WideResNet 28-10 as our Base model and we gather the results on the usual metrics for different number of heads:

Model	**Accuracy**	**NLL**	**Brier**	**ECE**	**MCE**
Base Model	95.4500	0.1866	0.0074	0.0272	0.8010
Multi-Output 3 heads	95.5160 ± 0.0609	0.1625 ± 0.0008	0.0071 ± 0.0000	0.0207 ± 0.0003	0.3808 ± 0.1791
Multi-Output 5 heads	95.5020 ± 0.0504	0.1550 ± 0.0012	0.0070 ± 0.0000	0.0175 ± 0.0005	0.3608 ± 0.2012
Multi-Output 7 heads	95.5140 ± 0.0467	0.1515 ± 0.0012	**0.0069 ± 0.0000**	0.0140 ± 0.0007	0.3380 ± 0.2342
Multi-Output 10 heads	**95.5400 ± 0.0477**	**0.1485 ± 0.0007**	**0.0069 ± 0.0000**	**0.0102 ± 0.0003**	**0.2441 ± 0.0138**

Table 5.1: Cifar-10 Standard results for each model (best result in **bold**, second-best result underlined).

We provide the same results on the corrupted version of the dataset.

Model	**cA**	**cNLL**	**cECE**	**cBrier**	**cMCE**
Base Model	73.4782	1.2986	0.1712	0.0422	0.3732
Multi-Output 3 heads	73.6795 ± 0.1242	1.0879 ± 0.0142	0.1463 ± 0.0025	0.0401 ± 0.0003	0.3256 ± 0.0089
Multi-Output 5 heads	73.7559 ± 0.0611	1.0303 ± 0.0082	0.1336 ± 0.0017	0.0392 ± 0.0001	0.2925 ± 0.0183
Multi-Output 7 heads	73.6704 ± 0.1611	0.9921 ± 0.0114	0.1228 ± 0.0033	0.0387 ± 0.0004	0.2969 ± 0.0198
Multi-Output 10 heads	**73.7653 ± 0.0515**	**0.9467 ± 0.0056**	**0.1114 ± 0.0013**	**0.0380 ± 0.0001**	**0.2607 ± 0.0160**

Table 5.2: Cifar-10-C Corrupted results for each model (best result in **bold**, second-best result underlined).

For the CIFAR-10 dataset we clearly observe that the 10 heads setting outperforms the others on every metrics even on the corrupted variants.

Since we did not have enough time to compare these results to the current baseline available at Google Research [2023] we will compare the relative improvements to the base model between our model and the baselines. To do so, we can apply the following formula:

$$\text{Relative Improvement (\%)} = \frac{\text{Metric}_{\text{Model}} - \text{Metric}_{\text{Base}}}{|\text{Metric}_{\text{Base}}|} \times 100$$

For values where improvement is positive (e.g., Accuracy), the formula measures the percentage increase. For metrics where lower is better (e.g., NLL, Brier score, ECE, MCE), the formula captures the percentage reduction.

We highlight that in the following table 5.3 of results the number of parameters for the Multi-Output models accounts for the parameters of the base-model which is 36,489,290 + the parameters of the module.

Multi-Output k heads models show modest improvements in accuracy (ranging from +0.05% to +0.09%), but they demonstrate significant improvements in calibration. The 10 heads model provides the largest reductions in NLL (-20.41%) and ECE (-62.50%). For corrupted data, the 10 heads model also performs best, with cNLL (-27.11%) and cECE (-34.94%). As a post-hoc method, Multi-Output achieves these improvements without requiring retraining and only uses one forward pass.

Baselines such as BatchEnsemble and SNGP with AugMix show better overall performance, particularly in accuracy (+0.94%) and calibration (NLL -35.22%, ECE -80.53%). SNGP-based methods also perform strongly on corrupted data, with SNGP with AugMix achieving cA (+13.07%) and cECE (-90.20%). However, these models require retraining and multiple forward passes, leading to increased computational cost.

In summary, the Multi-Output method provides smaller gains in accuracy but significant calibration improvements with minimal computational overhead, making it an efficient post-

Model	**Accuracy (%)**	**NLL (%)**	**ECE (%)**	**cA (%)**	**cNLL (%)**	**cECE (%)**	**Parameters**	**Forward Passes**
Multi-Output k heads compared to Base Model								
Multi-Output 3 heads	+0.07	-12.91	-23.90	+0.27	-16.23	-14.55	36.5M	1
Multi-Output 5 heads	+0.05	-16.95	-35.66	+0.38	-20.65	-21.98	36.5M	1
Multi-Output 7 heads	+0.07	-18.80	-48.53	+0.26	-23.57	-28.30	36.5M	1
Multi-Output 10 heads	+0.09	-20.41	-62.50	+0.39	-27.11	-34.94	36.6M	1
Baselines compared to Deterministic								
BatchEnsemble (size=4)	+0.31	-13.99	-23.37	+2.23	-7.62	-18.95	36.6M	4
Hyper-BatchEnsemble (size=4)	+0.31	-20.75	-60.17	-	-	-	73.1M	4
MIMO	+0.42	-22.01	-56.28	+0.66	-6.96	-26.80	36.5M	1
Rank-1 BNN (Gaussian, size=4)	+0.31	-19.87	-65.40	+0.79	-7.69	-47.71	36.6M	4
Rank-1 BNN (Cauchy, size=4)	+0.52	-24.53	-60.17	+4.41	-20.67	-41.18	36.6M	4
SNGP	+0.00	-15.72	-69.57	+2.42	-5.73	-49.02	37.5M	1
SNGP, with AugMix	**+0.94**	**-35.22**	**-80.53**	**+13.07**	-68.57	**-90.20**	37.5M	1
SNGP, with MC Dropout (size=10)	-0.10	-17.61	-65.29	+1.58	-7.62	-46.41	37.5M	10
SNGP, with BatchEnsemble (size=4)	+0.21	-20.12	-73.94	+2.00	-7.62	-47.71	37.5M	4
SNGP Ensemble (size=4)	+0.70	-31.45	-78.36	+3.39	-7.62	-51.63	150M	4
Monte Carlo Dropout (size=1)	-0.10	+0.62	+4.33	-7.31	+1.90	+8.50	36.5M	1
Monte Carlo Dropout (size=30)	+0.10	-8.81	-17.84	+1.58	-1.10	-9.77	36.5M	30
Monte Carlo Dropout, improved (size=30)	+0.21	-27.04	-78.40	+3.66	**-68.82**	-55.54	36.5M	30
Ensemble (size=4)	+0.63	-28.30	-56.72	+3.49	-7.62	-43.14	146M	4
Hyper-deep ensemble (size=4)	+0.63	-24.53	-60.17	+4.02	-7.62	-48.37	146M	4
Variational inference (sample=1)	-1.35	+32.70	+25.32	-6.32	+39.05	+18.30	73M	1
Posterior Network	-3.02	+126.42	+385.28	-1.18	+51.43	+33.51	36.6M	1

Table 5.3: Relative improvement: Multi-Output k heads models compared to the Base Model, and Baselines compared to the Deterministic model for Cifar-10 (best result in **bold**, second-best result underlined).

hoc alternative compared to more resource-intensive baselines.

5.2.2 Cifar-100

We study the same metrics for the Cifar-100 dataset and we compare the results:

Model	**Accuracy**	**NLL**	**Brier**	**ECE**	**MCE**
Base Model	78.1800	1.1427	0.0035	0.1342	0.3428
Multi-Output 3 heads	78.7780 ± 0.1607	0.8587 ± 0.0027	0.0031 ± 0.0000	0.0787 ± 0.0012	0.1914 ± 0.0090
Multi-Output 5 heads	**78.9300 ± 0.1384**	0.8198 ± 0.0037	**0.0030 ± 0.0000**	0.0570 ± 0.0009	0.1517 ± 0.0104
Multi-Output 7 heads	78.6940 ± 0.0726	0.8109 ± 0.0037	**0.0030 ± 0.0000**	0.0449 ± 0.0012	**0.1211 ± 0.0080**
Multi-Output 10 heads	78.5980 ± 0.1209	**0.8024 ± 0.0059**	**0.0030 ± 0.0000**	**0.0299 ± 0.0021**	0.2500 ± 0.3428

Table 5.4: Cifar-100 Standard results for each model (best result in **bold**, second-best result underlined).

The results reveal a trade-off between accuracy and calibration. In the standard CIFAR-100 setup, multi-output models improve both accuracy and calibration over the base model. The 5-head model achieves the highest accuracy, while the 10-head model exhibits the best calibration (NLL: 0.8024, ECE: 0.0299). This demonstrates that increasing the number of heads enhances calibration at the expense of minor accuracy drops.

Model	**cA**	**cNLL**	**cECE**	**cBrier**	**cMCE**
Base Model	44.9554	4.2648	0.3789	0.0089	0.5606
Multi-Output 3 heads	46.2547 ± 0.0901	3.0205 ± 0.0329	0.2558 ± 0.0029	0.0077 ± 0.0000	0.4238 ± 0.0122
Multi-Output 5 heads	**46.3800 ± 0.1247**	2.8274 ± 0.0252	0.2181 ± 0.0033	0.0075 ± 0.0000	0.3807 ± 0.0145
Multi-Output 7 heads	46.1950 ± 0.0900	2.7511 ± 0.0285	0.1950 ± 0.0037	0.0074 ± 0.0000	0.3438 ± 0.0091
Multi-Output 10 heads	46.2964 ± 0.1126	**2.6433 ± 0.0186**	**0.1629 ± 0.0031**	**0.0072 ± 0.0000**	**0.2929 ± 0.0037**

Table 5.5: Cifar-100-C Corrupted results for each model (best result in **bold**, second-best result underlined).

In the CIFAR-100-C setting, a similar pattern emerges. The 5-head model yields the highest accuracy (46.38%), but the 10-head model offers the best calibration (NLL: 2.6433, ECE: 0.1629). Thus, while the 5-head model strikes a balance, models with more heads provide superior uncertainty quantification, crucial in corrupted data scenarios.

We highlight here too that in the following table 5.6 of results the number of parameters for the Multi-Output models accounts for the parameters of the base-model which is also 36,546,980 + the parameters of the module.

Model	**Accuracy (%)**	**NLL (%)**	**ECE (%)**	**cA (%)**	**cNLL (%)**	**cECE (%)**	**Parameters**	**Forward Passes**
Multi-Output k heads compared to Base Model								
Multi-Output 3 heads	+0.77	-24.83	-41.34	+2.89	-29.16	-32.48	36.9M	1
Multi-Output 5 heads	+0.96	-28.30	-57.54	+3.17	-33.67	-42.42	37.4M	1
Multi-Output 7 heads	+0.66	-29.05	-66.54	+2.75	-35.46	-48.52	38.0M	1
Multi-Output 10 heads	+0.53	**-29.79**	-77.72	+2.98	-38.01	-56.99	39.2M	1
Baselines compared to Deterministic								
BatchEnsemble (size=4)	+2.63	-21.14	-69.08	+1.73	-5.19	-37.66	36.6M	4
Hyper-BatchEnsemble (size=4)	+2.63	-22.51	-76.70	-	-	-	36.6M	4
MIMO	+2.75	-21.14	-74.33	+2.33	+3.56	-46.03	36.5M	1
Rank-1 BNN (Gaussian, size=4)	+1.88	-20.91	-79.00	+2.43	+3.67	-51.05	36.6M	4
Rank-1 BNN (Cauchy, size=4)	+3.26	-21.20	-86.00	+6.41	-24.44	-40.59	36.6M	4
SNGP	+0.50	-7.80	-76.66	+0.50	-25.19	-61.34	37.5M	1
SNGP, with AugMix	+0.97	-5.91	-71.99	**+14.53**	**-52.63**	**-77.43**	37.5M	1
SNGP Ensemble (size=4)	+2.13	-24.00	**-87.15**	+5.43	-24.44	-62.10	150M	4
Monte Carlo Dropout (size=1)	-0.25	-0.94	-41.51	-7.74	+7.41	-15.93	36.5M	1
Ensemble (size=4)	+3.64	-23.89	-75.49	+2.90	-16.11	-43.84	146M	4
Hyper-deep ensemble (size=4)	**+4.02**	-25.31	-74.30	+3.00	-24.44	-46.44	146M	4
Variational inference (sample=1)	-2.51	+7.89	+13.88	-8.13	+17.78	+13.39	73M	1
Heteroscedastic	+0.50	-5.14	-31.12	+0.24	-2.88	-25.73	37M	1
Heteroscedastic Ensemble (size=4)	+2.38	-23.67	-69.65	+1.75	-7.38	-56.89	148M	4

Table 5.6: Relative improvement: Multi-Output k heads models compared to the Base Model, and Baselines compared to the Deterministic model for Cifar-100 (best result in **bold**, second-best result underlined).

The Multi-Output models show moderate improvements in accuracy, with the 5 heads model achieving the highest at +0.96%. The most significant gains for Multi-Output models are in calibration, particularly with the 10 heads model, which shows the largest reductions in NLL (-29.79%) and ECE (-77.72%). For corrupted data, the 10 heads model again shows substantial improvement, with cNLL (-38.01%) and cECE (-56.99%). These gains are achieved without retraining and only require one forward pass.

In contrast, Baselines such as BatchEnsemble and SNGP with AugMix provide stronger overall performance, particularly in accuracy and calibration on corrupted data. SNGP with AugMix achieves a substantial improvement in cA (+14.53%) and cECE (-77.43%). However, again, these models require retraining and multiple forward passes, significantly increasing their computational cost. Hyper-deep Ensemble also performs well with the highest accuracy gain (+4.02%) but requires 146M parameters and four forward passes.

These results highlight the strength of our method in its simplicity and post-hoc, model-agnostic setting, as it achieves substantial calibration gains with minimal computational overhead compared to the more complex alternatives requiring retraining.

5.2.3 MNIST

We study the same metrics for the MNIST dataset and we compare the results:

Model	**Accuracy**	**NLL**	**Brier**	**ECE**	**MCE**
Base Model	98.4900	0.0517	0.0023	0.0060	0.6878
Multi-Output 3 heads	98.7020 ± 0.0299	0.0417 ± 0.0007	**0.0020 ± 0.0000**	0.0049 ± 0.0008	0.4398 ± 0.1431
Multi-Output 5 heads	**98.7140 ± 0.0383**	**0.0404 ± 0.0006**	**0.0020 ± 0.0000**	0.0044 ± 0.0008	0.3611 ± 0.0660
Multi-Output 7 heads	98.6560 ± 0.0383	0.0432 ± 0.0007	0.0021 ± 0.0000	0.0030 ± 0.0005	**0.3508 ± 0.1526**
Multi-Output 10 heads	98.6440 ± 0.0361	0.0426 ± 0.0008	0.0021 ± 0.0000	0.0035 ± 0.0004	0.4666 ± 0.1921
Multi-Output 13 heads	98.6020 ± 0.0426	0.0448 ± 0.0018	0.0021 ± 0.0001	0.0032 ± 0.0005	0.4778 ± 0.1772
Multi-Output 15 heads	98.5980 ± 0.0337	0.0453 ± 0.0009	0.0022 ± 0.0001	**0.0023 ± 0.0004**	0.4130 ± 0.2155

Table 5.7: MNIST Calibration results for each model (best result in **bold**, second-best result underlined).

The Multi-Output k heads models show moderate improvements in accuracy and strong reductions in NLL and ECE compared to the base model. The 15 heads model performs best in ECE, with a reduction of 61.67%, making it the most effective in calibration improvement while maintaining a rather lightweight structure with 102.5K parameters and requiring a single forward pass.

In contrast, the Ensemble (size=10) baseline demonstrates a substantial improvement in NLL with a 56.76% reduction and a smaller improvement in ECE at 16.67%, despite its larger parameter count (600K) and higher computational cost (10 forward passes). Other baselines, such as Refined VI and Variational Inference, show less significant improvements.

Model	Accuracy (%)	NLL (%)	ECE (%)	Parameters	Forward Passes
Multi-Output k heads compared to Base Model					
Multi-Output 3 heads	+0.21	-19.33	-18.33	48.8K	1
Multi-Output 5 heads	+0.23	-21.85	-26.67	53.8K	1
Multi-Output 7 heads	+0.17	-16.43	-50.00	60.3K	1
Multi-Output 10 heads	+0.16	-17.62	-41.67	73.1K	1
Multi-Output 13 heads	+0.11	-13.34	-46.67	89.5K	1
Multi-Output 15 heads	+0.11	-12.87	**-61.67**	102.5K	1
Baselines compared to Deterministic (using Base Model ECE)					
Ensemble (size=10)	**+0.30**	**-56.76**	-16.67	600K	10
Refined VI	-0.10	-8.11	-13.04	-	1
Variational Inference	-0.01	-6.28	-13.04	-	1

Table 5.8: Relative improvement: Multi-Output k heads models compared to the Base Model, and Baselines compared to the Deterministic model for MNIST, with number of parameters and forward passes.

5.3 OOD Experimental results

To explain our OOD experiments we first propose to understand this small experiment setting: After training a base model (LeNet5) on the MNIST dataset, we take a digit (6 in this case) and rotate it with increased degrees. The entropy of the base model's predictions, shown in blue, reflects its confidence in the outputs. Notably, the model exhibits overconfidence,

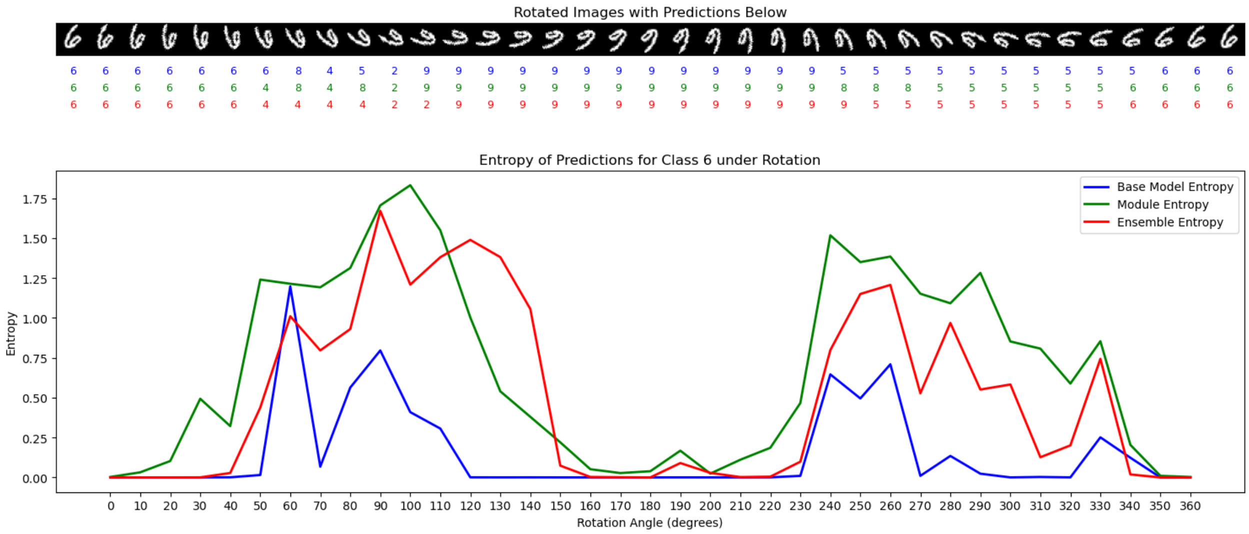


Figure 5.1: Evolution of the outputs and the entropy of the results with an increasing rotation of a 6 on the MNIST dataset with the LeNet5 as a base model

particularly between +60-100 degrees and +240-340 degrees where the entropy is higher but still too low on average, even when it incorrectly predicts the digit. In the middle range, the model identifies the rotated 6 as a 9, leading to increased confidence. The Ensemble of base models (in red) effectively reduces this overconfidence, displaying higher entropy when the predictions are incorrect. Similarly, our multi-output model with 5 heads (in green) also mitigates overconfidence in these scenarios.

In the next two images 5.2 and 5.3, we observe the differences in entropy (confidence) of the softmax outputs between in-distribution (ID) data (MNIST dataset) and out-of-distribution (OOD) data (FashionMNIST dataset). It is evident that the base model remains overly confident on certain OOD images, while the ensemble and the module better adjust by increasing uncertainty for those cases. We also note the similarity between the behaviors of the outputs of the Deep Ensemble and the module which was what was originally intended.

In 5.1, we could have used other metrics besides entropy to measure confidence in OOD detection, such as Maximum Softmax Probability (MSP) and Energy score. Introduced by Hendrycks and Gimpel [2017], MSP measures confidence based on the maximum value of the softmax output:

$$\mathrm{MSP}(x) = \max_k \frac{e^{f_k(x)}}{\sum_{j=1}^{K} e^{f_j(x)}}$$

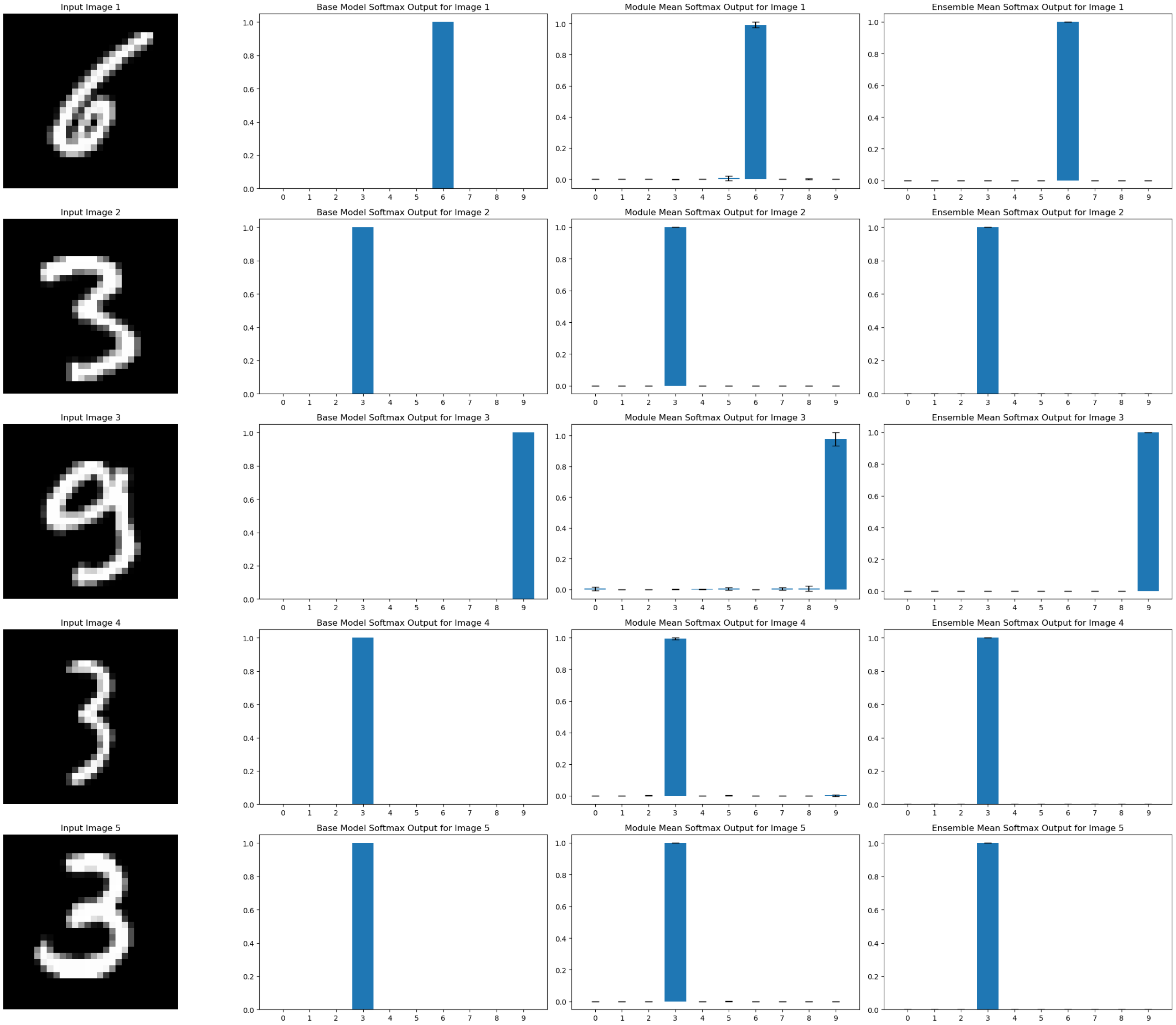
with $f_k(x)$ representing the output logits. A lower MSP score suggests higher uncertainty, indicating potential OOD samples. On the other hand, the Energy score, proposed by Liu et al. [2020b], directly computes the energy from the logits without softmax, avoiding overconfidence issues:

$$E(x) = -\log \sum_{k=1}^{K} e^{f_k(x)}$$

The Energy score is lower for in-distribution data and higher for OOD data, making it a robust confidence measure.

In OOD detection, Entropy, MSP and Energy scores are used to distinguish in-distribution data from out-of-distribution data. Given an input x, a lower MSP or higher Energy or Entropy score typically indicates that the input is OOD, as the model is less confident in its prediction. For detection, we set a threshold for these scores: inputs with scores below or above the threshold are flagged as OOD, while the rest are classified as ID. This simple thresholding mechanism effectively separates ID from OOD data.

Then, to assess how well Entropy, MSP and Energy scores differentiate ID from OOD data, we use metrics like the Area Under the Receiver Operating Characteristic curve (AUROC),



Input Image 1

Base Model Softmax Output for Image 1

Module Mean Softmax Output for Image 1

Ensemble Mean Softmax Output for Image 1

Input Image 2

Base Model Softmax Output for Image 2

Module Mean Softmax Output for Image 2

Ensemble Mean Softmax Output for Image 2

Input Image 3

Base Model Softmax Output for Image 3

Module Mean Softmax Output for Image 3

Ensemble Mean Softmax Output for Image 3

Input Image 4

Base Model Softmax Output for Image 4

Module Mean Softmax Output for Image 4

Ensemble Mean Softmax Output for Image 4

Input Image 5

Base Model Softmax Output for Image 5

Module Mean Softmax Output for Image 5

Ensemble Mean Softmax Output for Image 5

Figure 5.2: Softmax outputs of the different models in the case of ID (in distribution) data, base model on the left, module in the center and the ensemble on the right, along with the variance of each outputs.

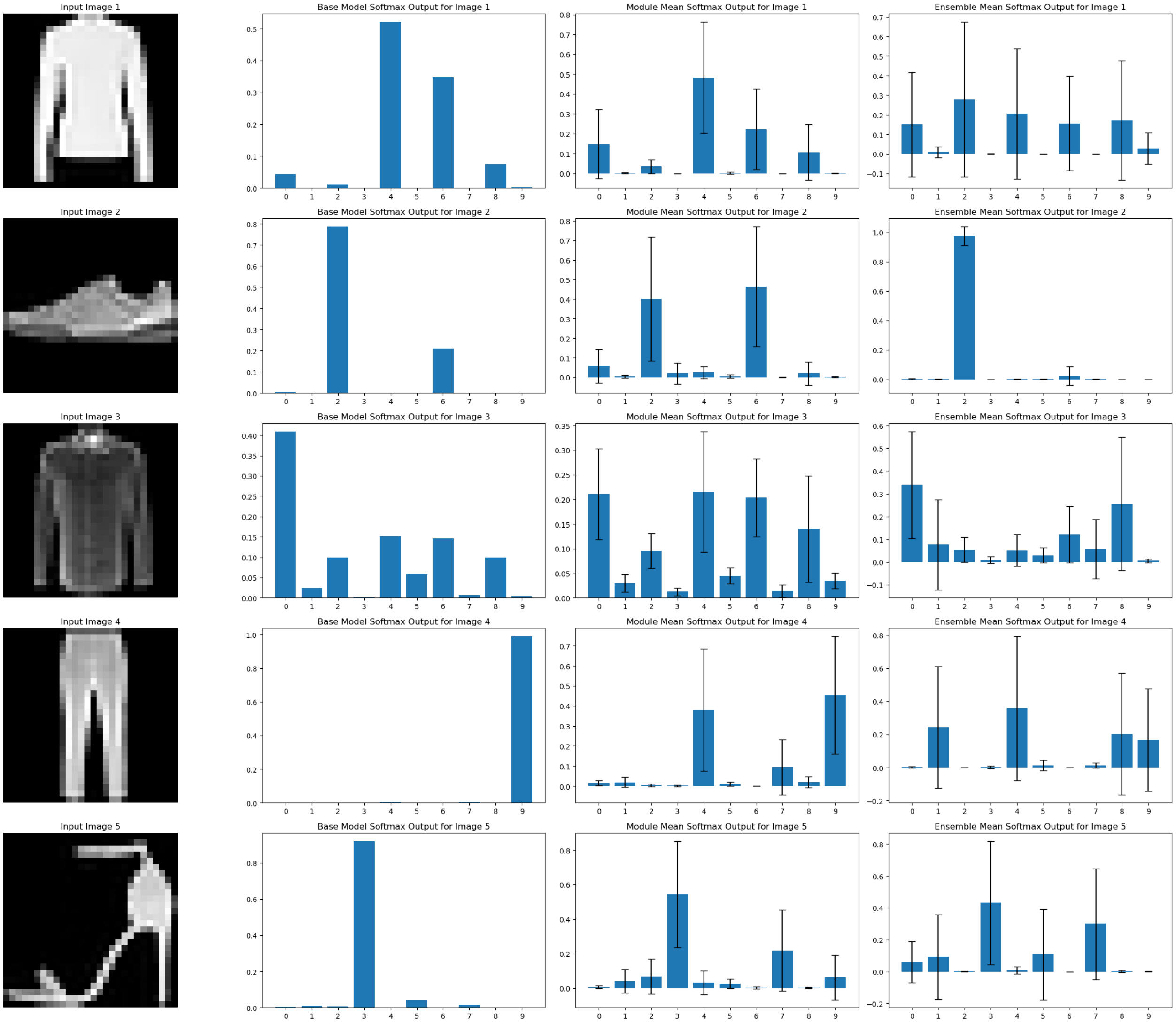


Figure 5.3: Softmax outputs of the different models in the case of OOD (out of distribution) data, base model on the left, module in the center and the ensemble on the right, along with the variance of each outputs.

Area Under the Precision-Recall curve (AUPR), and Area Under the Risk-coverage curve (AUR).

AUROC (Area Under the Receiver Operating Characteristic curve): This metric measures the ability of a model to distinguish between in-distribution (ID) and out-of-distribution (OOD) data. It is calculated by plotting the true positive rate (TPR) against the false positive rate (FPR) at various classification thresholds. An AUROC score of 1.0 represents perfect classification, while a score of 0.5 indicates random guessing. In the context of OOD detection, a high AUROC score signifies that the model is effectively able to separate ID and OOD samples based on uncertainty estimates or other measures.

AUPR (Area Under the Precision-Recall curve): This metric emphasizes the model's performance in identifying positive cases (OOD samples) when the data is imbalanced. AUPR is particularly useful in situations where the OOD samples are much less frequent compared to ID samples. A high AUPR score indicates that the model can successfully flag OOD examples without misclassifying too many ID examples as OOD. The score results can be interpreted in the same way as the AUROC score.

AUR (Area Under the Risk curve): This metric measures the trade-off between model risk and the fraction of OOD samples identified. It provides insight into the risk of making incorrect predictions as more OOD samples are detected. A lower AUR score is preferable as it indicates a more reliable model with lower associated risks when handling OOD data.

These metrics are computed across multiple OOD datasets to provide a comprehensive evaluation of model robustness. The datasets used include MNIST, FashionMNIST, CIFAR-10, CIFAR-100, SVHN, and TinyImageNet.

For CIFAR-10 on MNIST and FashionMNIST, the base model performs better overall, and there is no significant improvement from the Multi-Output module. In these cases, the Multi-Output models do not provide a meaningful enhancement in out-of-distribution detection.

However, for other datasets, such as SVHN and TinyImageNet, there is some improvement with the Multi-Output models, though the gains are not substantial. The Multi-Output models do offer some advantage with different head settings, particularly on more challenging datasets like CIFAR-100.

For CIFAR-100, the results show a noticeable improvement across the datasets, with the Multi-Output models demonstrating global enhancements in OOD detection with varying head configurations.

Models	**Metric**	**MNIST**			**FashionMNIST**			**CIFAR-100**			**SVHN**			**TinyImageNet**		
		AUROC	AUPR	AUR	AUROC	AUPR	AUR	AUROC	AUPR	AUR	AUROC	AUPR	AUR	AUROC	AUPR	AUR
Base Model	Entropy	0.9426	0.9267	0.4267	0.9552	0.9432	0.4432	0.9011	0.8837	0.3837	0.9349	0.9623	0.2399	0.8936	0.8827	0.3827
	MSP	0.9384	0.9155	**0.4155**	0.9499	0.9309	**0.4309**	0.8985	0.8748	**0.3748**	0.9311	0.9578	0.2353	0.8904	0.8725	**0.3725**
	Energy	0.9497	**0.9385**	0.4267	**0.9698**	**0.9623**	0.4623	0.8980	0.8928	0.3928	0.9475	**0.9699**	0.2474	0.8997	0.9003	0.4003
MO 3 heads	Entropy	0.9447 ± 0.0015	0.9296 ± 0.0021	0.4296 ± 0.0021	0.9584 ± 0.0011	0.9473 ± 0.0020	0.4473 ± 0.0020	0.9026 ± 0.0010	0.8890 ± 0.0007	0.3890 ± 0.0007	0.9371 ± 0.0045	0.9635 ± 0.0025	0.2410 ± 0.0025	0.8966 ± 0.0007	0.8901 ± 0.0003	0.3901 ± 0.0003
	MSP	0.9396 ± 0.0012	0.9190 ± 0.0013	0.4190 ± 0.0013	0.9521 ± 0.0009	0.9354 ± 0.0018	0.4354 ± 0.0018	0.8992 ± 0.0010	0.8792 ± 0.0011	0.3792 ± 0.0011	0.9323 ± 0.0048	0.9589 ± 0.0028	0.2364 ± 0.0028	0.8924 ± 0.0007	0.8789 ± 0.0006	0.3789 ± 0.0006
	Energy	0.9485 ± 0.0073	0.9266 ± 0.0139	0.4266 ± 0.0139	0.9672 ± 0.0048	0.9535 ± 0.0079	0.4535 ± 0.0079	0.9095 ± 0.0025	0.8972 ± 0.0030	0.3972 ± 0.0030	0.9443 ± 0.0074	0.9638 ± 0.0054	0.2413 ± 0.0054	0.9062 ± 0.0015	0.9026 ± 0.0021	0.4026 ± 0.0021
MO 5 heads	Entropy	0.9457 ± 0.0012	0.9322 ± 0.0013	0.4322 ± 0.0013	0.9613 ± 0.0015	0.9516 ± 0.0026	0.4516 ± 0.0026	0.9052 ± 0.0006	0.8926 ± 0.0005	0.3926 ± 0.0005	0.9346 ± 0.0031	0.9626 ± 0.0020	0.2401 ± 0.0020	0.8986 ± 0.0006	0.8935 ± 0.0005	0.3935 ± 0.0005
	MSP	0.9402 ± 0.0014	0.9210 ± 0.0014	0.4210 ± 0.0014	0.9543 ± 0.0010	0.9391 ± 0.0019	0.4391 ± 0.0019	0.9014 ± 0.0006	0.8828 ± 0.0008	0.3828 ± 0.0008	0.9291 ± 0.0028	0.9575 ± 0.0018	**0.2351 ± 0.0018**	0.8940 ± 0.0006	0.8821 ± 0.0007	0.3821 ± 0.0007
	Energy	0.9513 ± 0.0042	0.9327 ± 0.0099	0.4327 ± 0.0099	0.9678 ± 0.0036	0.9531 ± 0.0064	0.4531 ± 0.0064	0.9116 ± 0.0032	**0.8993 ± 0.0027**	0.3993 ± 0.0027	0.9423 ± 0.0098	0.9624 ± 0.0071	0.2399 ± 0.0071	0.9072 ± 0.0018	**0.9031 ± 0.0014**	0.4031 ± 0.0014
MO 7 heads	Entropy	0.9471 ± 0.0013	0.9330 ± 0.0015	0.4330 ± 0.0015	0.9619 ± 0.0005	0.9519 ± 0.0012	0.4519 ± 0.0012	0.9050 ± 0.0011	0.8929 ± 0.0014	0.3929 ± 0.0014	0.9382 ± 0.0044	0.9647 ± 0.0023	0.2422 ± 0.0023	**0.9082 ± 0.0011**	0.9025 ± 0.0016	0.4025 ± 0.0016
	MSP	0.9415 ± 0.0010	0.9224 ± 0.0014	0.4224 ± 0.0014	0.9549 ± 0.0003	0.9398 ± 0.0006	0.4398 ± 0.0006	0.9013 ± 0.0010	0.8831 ± 0.0011	0.3831 ± 0.0011	0.9323 ± 0.0046	0.9597 ± 0.0026	0.2372 ± 0.0026	0.8944 ± 0.0011	0.8830 ± 0.0010	0.3830 ± 0.0010
	Energy	0.9537 ± 0.0029	0.9332 ± 0.0072	0.4332 ± 0.0072	0.9685 ± 0.0039	0.9543 ± 0.0082	0.4543 ± 0.0082	0.9119 ± 0.0027	0.8975 ± 0.0038	0.3975 ± 0.0038	0.9447 ± 0.0035	0.9636 ± 0.0027	0.2411 ± 0.0027	0.8992 ± 0.0013	0.8944 ± 0.0014	0.3944 ± 0.0014
MO 10 heads	Entropy	0.9487 ± 0.0011	0.9359 ± 0.0018	0.4359 ± 0.0018	0.9628 ± 0.0014	0.9531 ± 0.0030	0.4531 ± 0.0030	0.9060 ± 0.0008	0.8948 ± 0.0006	0.3948 ± 0.0006	0.9379 ± 0.0023	0.9644 ± 0.0020	0.2419 ± 0.0020	0.9004 ± 0.0007	0.8966 ± 0.0007	0.3966 ± 0.0007
	MSP	0.9427 ± 0.0009	0.9248 ± 0.0013	0.4248 ± 0.0013	0.9552 ± 0.0010	0.9403 ± 0.0024	0.4403 ± 0.0024	0.9018 ± 0.0007	0.8845 ± 0.0006	0.3845 ± 0.0006	0.9314 ± 0.0016	0.9591 ± 0.0015	0.2367 ± 0.0015	0.8951 ± 0.0006	0.8845 ± 0.0007	0.3845 ± 0.0007
	Energy	**0.9552 ± 0.0046**	0.9330 ± 0.0105	0.4330 ± 0.0105	0.9674 ± 0.0039	0.9510 ± 0.0091	0.4510 ± 0.0091	**0.9145 ± 0.0014**	0.8982 ± 0.0023	0.3982 ± 0.0023	**0.9477 ± 0.0066**	0.9638 ± 0.0052	0.2413 ± 0.0052	0.9076 ± 0.0019	0.8999 ± 0.0026	0.3999 ± 0.0026

Table 5.9: OOD Metrics for CIFAR-10 across various datasets and models. Best results are in **bold**, second-best are underlined. For AUR, the best result is the lowest value.

Models	Metric	MNIST			FashionMNIST			CIFAR-10			SVHN			TinyImageNet		
		AUROC	AUPR	AUR	AUROC	AUPR	AUR	AUROC	AUPR	AUR	AUROC	AUPR	AUR	AUROC	AUPR	AUR
Base Model	Entropy	0.7026	0.6560	0.1560	0.8347	0.7877	0.2877	0.7930	0.7502	0.2502	0.5842	0.7823	0.0598	0.7996	0.7641	0.2641
	MSP	0.6982	0.6528	0.1528	0.8218	0.7688	0.2688	0.7849	0.7366	0.2366	0.5779	0.7755	**0.0530**	0.7897	0.7464	0.2464
	Energy	0.7127	0.6577	0.1577	**0.8841**	**0.8344**	0.3344	0.8033	**0.7659**	0.2659	0.6317	0.8084	0.0859	**0.8238**	**0.7922**	0.2922
MO 3 heads	Entropy	0.7171 ± 0.0112	0.6637 ± 0.0093	0.1637 ± 0.0093	0.8667 ± 0.0045	0.8231 ± 0.0067	0.3231 ± 0.0067	0.8053 ± 0.0017	0.7626 ± 0.0022	0.2626 ± 0.0022	0.6765 ± 0.0120	0.8348 ± 0.0079	0.1123 ± 0.0079	0.8173 ± 0.0007	0.7865 ± 0.0007	0.2865 ± 0.0007
	MSP	0.7105 ± 0.0105	0.6635 ± 0.0079	0.1635 ± 0.0079	0.8471 ± 0.0040	0.8024 ± 0.0048	0.3024 ± 0.0048	0.7974 ± 0.0015	0.7541 ± 0.0015	0.2541 ± 0.0015	0.6638 ± 0.0111	0.8243 ± 0.0070	0.1018 ± 0.0070	0.8049 ± 0.0007	0.7697 ± 0.0012	0.2697 ± 0.0012
	Energy	0.7212 ± 0.0139	0.6390 ± 0.0138	0.1390 ± 0.0138	0.8641 ± 0.0046	0.7956 ± 0.0083	0.2956 ± 0.0083	0.7806 ± 0.0052	0.7300 ± 0.0049	0.2300 ± 0.0049	0.6847 ± 0.0121	0.8141 ± 0.0087	0.0916 ± 0.0087	0.8190 ± 0.0024	0.7804 ± 0.0043	0.2804 ± 0.0043
MO 5 heads	Entropy	0.7281 ± 0.0121	0.6700 ± 0.0105	0.1700 ± 0.0105	0.8663 ± 0.0037	0.8187 ± 0.0055	0.3187 ± 0.0055	0.8056 ± 0.0032	0.7624 ± 0.0044	0.2624 ± 0.0044	0.7070 ± 0.0139	0.8483 ± 0.0074	0.1258 ± 0.0074	0.8203 ± 0.0007	0.7892 ± 0.0007	0.2892 ± 0.0007
	MSP	0.7208 ± 0.0112	0.6691 ± 0.0094	0.1691 ± 0.0094	0.8464 ± 0.0032	0.7989 ± 0.0049	0.2989 ± 0.0049	0.7982 ± 0.0027	0.7549 ± 0.0043	0.2549 ± 0.0043	0.6932 ± 0.0134	0.8377 ± 0.0070	0.1152 ± 0.0070	0.8080 ± 0.0007	0.7733 ± 0.0012	0.2733 ± 0.0012
	Energy	0.7236 ± 0.0147	0.6363 ± 0.0130	0.1363 ± 0.0130	0.8430 ± 0.0113	0.7668 ± 0.0165	0.2668 ± 0.0165	0.7647 ± 0.0070	0.7102 ± 0.0058	0.2102 ± 0.0058	0.7120 ± 0.0170	0.8266 ± 0.0101	0.1041 ± 0.0101	0.8101 ± 0.0021	0.7651 ± 0.0029	0.2651 ± 0.0029
MO 7 heads	Entropy	**0.7308 ± 0.0133**	**0.6713 ± 0.0130**	0.1713 ± 0.0130	0.8658 ± 0.0035	0.8181 ± 0.0059	0.3181 ± 0.0059	0.8071 ± 0.0018	0.7629 ± 0.0022	0.2629 ± 0.0022	0.7048 ± 0.0159	0.8492 ± 0.0087	0.1268 ± 0.0087	0.8212 ± 0.0004	0.7894 ± 0.0006	0.2894 ± 0.0006
	MSP	0.7223 ± 0.0128	0.6691 ± 0.0116	0.1691 ± 0.0116	0.8467 ± 0.0029	0.7994 ± 0.0049	0.2994 ± 0.0049	0.8000 ± 0.0015	0.7561 ± 0.0024	0.2561 ± 0.0024	0.6896 ± 0.0151	0.8375 ± 0.0082	0.1151 ± 0.0082	0.8094 ± 0.0005	0.7746 ± 0.0012	0.2746 ± 0.0012
	Energy	0.7220 ± 0.0131	0.6319 ± 0.0104	0.1319 ± 0.0104	0.8199 ± 0.0094	0.7347 ± 0.0114	0.2347 ± 0.0114	0.7611 ± 0.0040	0.7023 ± 0.0034	0.2023 ± 0.0034	0.6961 ± 0.0144	0.8179 ± 0.0065	0.0954 ± 0.0065	0.8028 ± 0.0020	0.7554 ± 0.0020	0.2554 ± 0.0020
MO 10 heads	Entropy	0.7202 ± 0.0112	0.6576 ± 0.0108	0.1576 ± 0.0108	0.8675 ± 0.0036	0.8175 ± 0.0048	0.3175 ± 0.0048	**0.8082 ± 0.0013**	0.7631 ± 0.0022	0.2631 ± 0.0022	**0.7161 ± 0.0089**	**0.8533 ± 0.0040**	0.1308 ± 0.0040	0.8231 ± 0.0008	0.7908 ± 0.0009	0.2908 ± 0.0009
	MSP	0.7144 ± 0.0100	0.6601 ± 0.0088	0.1601 ± 0.0088	0.8482 ± 0.0025	0.7997 ± 0.0049	0.2997 ± 0.0049	0.8018 ± 0.0011	0.7575 ± 0.0019	0.2575 ± 0.0019	0.7007 ± 0.0091	0.8418 ± 0.0040	0.1193 ± 0.0040	0.8116 ± 0.0007	0.7764 ± 0.0010	0.2764 ± 0.0010
	Energy	0.7073 ± 0.0120	0.6190 ± 0.0104	**0.1190 ± 0.0104**	0.8089 ± 0.0028	0.7228 ± 0.0031	**0.2228 ± 0.0031**	0.7444 ± 0.0056	0.6850 ± 0.0046	**0.1850 ± 0.0046**	0.6954 ± 0.0122	0.8165 ± 0.0061	0.0940 ± 0.0061	0.7938 ± 0.0035	0.7438 ± 0.0042	**0.2438 ± 0.0042**

Table 5.10: OOD Metrics for CIFAR-100 across various datasets and models. Best results are in **bold**, second-best are underlined. For AUR, the best result is the lowest value.

Models	**Metric**	**FashionMNIST**			**SVHN**			**CIFAR-10**			**CIFAR-100**			**TinyImageNet**		
		AUROC	**AUPR**	**AUR**	**AUROC**	**AUPR**	**AUR**	**AUROC**	**AUPR**	**AUR**	**AUROC**	**AUPR**	**AUR**	**AUROC**	**AUPR**	**AUR**
Base Model	Entropy	0.9356	0.9319	0.4319	0.9748	0.9890	0.2666	0.9561	0.9519	0.4519	0.9587	0.9561	0.4561	0.9614	0.9577	0.4577
	MSP	0.9333	0.9256	0.4256	0.9716	0.9863	0.2638	0.9534	0.9447	**0.4447**	0.9558	0.9490	**0.4490**	0.9585	0.9505	**0.4505**
	Energy	0.8993	0.9039	**0.4039**	0.9775	0.9905	0.2680	0.9560	0.9564	0.4564	0.9590	0.9606	0.4606	0.9645	0.9641	0.4641
MO 3 heads	Entropy	0.9342 ± 0.0065	0.9332 ± 0.0056	0.4332 ± 0.0056	0.9796 ± 0.0016	0.9914 ± 0.0007	0.2690 ± 0.0007	0.9682 ± 0.0009	0.9659 ± 0.0012	0.4659 ± 0.0012	0.9709 ± 0.0010	0.9696 ± 0.0014	0.4696 ± 0.0014	0.9738 ± 0.0013	0.9719 ± 0.0016	0.4719 ± 0.0016
	MSP	0.9327 ± 0.0065	0.9287 ± 0.0059	0.4287 ± 0.0059	0.9771 ± 0.0015	0.9896 ± 0.0006	0.2671 ± 0.0006	0.9658 ± 0.0009	0.9605 ± 0.0012	0.4605 ± 0.0012	0.9684 ± 0.0009	0.9643 ± 0.0014	0.4643 ± 0.0014	0.9712 ± 0.0012	0.9665 ± 0.0017	0.4665 ± 0.0017
	Energy	0.9323 ± 0.0211	0.9307 ± 0.0220	0.4307 ± 0.0220	0.9713 ± 0.0053	0.9845 ± 0.0043	0.2621 ± 0.0043	0.9647 ± 0.0041	0.9584 ± 0.0072	0.4584 ± 0.0072	0.9665 ± 0.0052	0.9602 ± 0.0086	0.4602 ± 0.0086	0.9687 ± 0.0047	0.9615 ± 0.0089	0.4615 ± 0.0089
MO 5 heads	Entropy	0.9420 ± 0.0015	**0.9417 ± 0.0017**	0.4417 ± 0.0017	0.9809 ± 0.0014	0.9921 ± 0.0005	0.2696 ± 0.0005	0.9676 ± 0.0013	0.9653 ± 0.0014	0.4653 ± 0.0014	0.9705 ± 0.0012	0.9694 ± 0.0013	0.4694 ± 0.0013	0.9732 ± 0.0014	0.9714 ± 0.0016	0.4714 ± 0.0016
	MSP	0.9404 ± 0.0014	0.9374 ± 0.0018	0.4374 ± 0.0018	0.9785 ± 0.0013	0.9903 ± 0.0005	0.2678 ± 0.0005	0.9652 ± 0.0012	0.9598 ± 0.0014	0.4598 ± 0.0014	0.9680 ± 0.0012	0.9639 ± 0.0013	0.4639 ± 0.0013	0.9706 ± 0.0013	0.9660 ± 0.0016	0.4660 ± 0.0016
	Energy	0.9354 ± 0.0163	0.9362 ± 0.0159	0.4362 ± 0.0159	0.9768 ± 0.0034	0.9878 ± 0.0030	0.2653 ± 0.0030	0.9676 ± 0.0038	0.9625 ± 0.0064	0.4625 ± 0.0064	0.9697 ± 0.0034	0.9651 ± 0.0063	0.4651 ± 0.0063	0.9729 ± 0.0035	0.9670 ± 0.0067	0.4670 ± 0.0067
MO 7 heads	Entropy	0.9330 ± 0.0061	0.9326 ± 0.0053	0.4326 ± 0.0053	0.9797 ± 0.0010	0.9916 ± 0.0004	0.2691 ± 0.0004	0.9670 ± 0.0012	0.9649 ± 0.0011	0.4649 ± 0.0011	0.9699 ± 0.0011	0.9689 ± 0.0010	0.4689 ± 0.0010	0.9731 ± 0.0011	0.9714 ± 0.0010	0.4714 ± 0.0010
	MSP	0.9310 ± 0.0061	0.9276 ± 0.0053	0.4276 ± 0.0053	0.9767 ± 0.0011	0.9895 ± 0.0004	0.2671 ± 0.0004	0.9641 ± 0.0012	0.9591 ± 0.0009	0.4591 ± 0.0009	0.9669 ± 0.0011	0.9630 ± 0.0009	0.4630 ± 0.0009	0.9700 ± 0.0010	0.9657 ± 0.0009	0.4657 ± 0.0009
	Energy	0.9376 ± 0.0165	0.9392 ± 0.0156	0.4392 ± 0.0156	0.9771 ± 0.0024	0.9875 ± 0.0020	0.2650 ± 0.0020	**0.9725 ± 0.0017**	**0.9676 ± 0.0027**	0.4676 ± 0.0027	**0.9735 ± 0.0015**	0.9689 ± 0.0024	0.4689 ± 0.0024	**0.9768 ± 0.0016**	0.9715 ± 0.0027	0.4715 ± 0.0027
MO 10 heads	Entropy	0.9327 ± 0.0053	0.9317 ± 0.0051	0.4317 ± 0.0051	0.9799 ± 0.0005	0.9917 ± 0.0002	0.2692 ± 0.0002	0.9680 ± 0.0008	0.9663 ± 0.0009	0.4663 ± 0.0009	0.9710 ± 0.0010	0.9703 ± 0.0011	0.4703 ± 0.0011	0.9741 ± 0.0011	**0.9728 ± 0.0012**	0.4728 ± 0.0012
	MSP	0.9307 ± 0.0053	0.9267 ± 0.0053	0.4267 ± 0.0053	0.9769 ± 0.0005	0.9897 ± 0.0003	0.2673 ± 0.0003	0.9652 ± 0.0007	0.9605 ± 0.0010	0.4605 ± 0.0010	0.9681 ± 0.0010	0.9647 ± 0.0012	0.4647 ± 0.0012	0.9711 ± 0.0011	0.9671 ± 0.0013	0.4671 ± 0.0013
	Energy	0.9307 ± 0.0232	0.9305 ± 0.0262	0.4305 ± 0.0262	0.9681 ± 0.0142	0.9796 ± 0.0118	0.2571 ± 0.0118	0.9680 ± 0.0100	0.9581 ± 0.0170	0.4581 ± 0.0170	0.9694 ± 0.0100	0.9589 ± 0.0178	0.4589 ± 0.0178	0.9713 ± 0.0109	0.9600 ± 0.0193	0.4600 ± 0.0193
MO 13 heads	Entropy	0.9383 ± 0.0061	0.9367 ± 0.0057	0.4367 ± 0.0057	0.9801 ± 0.0008	0.9918 ± 0.0003	0.2693 ± 0.0003	0.9684 ± 0.0020	0.9665 ± 0.0018	0.4665 ± 0.0018	0.9714 ± 0.0018	**0.9705 ± 0.0016**	0.4705 ± 0.0016	0.9741 ± 0.0015	**0.9728 ± 0.0014**	0.4728 ± 0.0014
	MSP	0.9364 ± 0.0061	0.9316 ± 0.0058	0.4316 ± 0.0058	0.9769 ± 0.0009	0.9897 ± 0.0003	0.2672 ± 0.0003	0.9653 ± 0.0020	0.9602 ± 0.0020	0.4602 ± 0.0020	0.9682 ± 0.0019	0.9642 ± 0.0018	0.4642 ± 0.0018	0.9709 ± 0.0016	0.9666 ± 0.0016	0.4666 ± 0.0016
	Energy	**0.9428 ± 0.0175**	0.9414 ± 0.0187	0.4414 ± 0.0187	0.9647 ± 0.0102	0.9760 ± 0.0091	**0.2536 ± 0.0091**	0.9637 ± 0.0065	0.9506 ± 0.0109	0.4506 ± 0.0109	0.9651 ± 0.0071	0.9507 ± 0.0126	0.4507 ± 0.0126	0.9665 ± 0.0075	0.9515 ± 0.0134	0.4515 ± 0.0134
MO 15 heads	Entropy	0.9351 ± 0.0059	0.9341 ± 0.0058	0.4341 ± 0.0058	**0.9811 ± 0.0005**	**0.9922 ± 0.0002**	0.2698 ± 0.0002	0.9676 ± 0.0004	0.9659 ± 0.0004	0.4659 ± 0.0004	0.9710 ± 0.0004	0.9703 ± 0.0003	0.4703 ± 0.0003	0.9739 ± 0.0003	0.9727 ± 0.0004	0.4727 ± 0.0004
	MSP	0.9331 ± 0.0062	0.9293 ± 0.0065	0.4293 ± 0.0065	0.9779 ± 0.0006	0.9902 ± 0.0003	0.2678 ± 0.0003	0.9643 ± 0.0005	0.9597 ± 0.0005	0.4597 ± 0.0005	0.9676 ± 0.0005	0.9642 ± 0.0005	0.4642 ± 0.0005	0.9704 ± 0.0003	0.9666 ± 0.0004	0.4666 ± 0.0004
	Energy	0.9307 ± 0.0229	0.9335 ± 0.0210	0.4335 ± 0.0210	0.9675 ± 0.0079	0.9781 ± 0.0077	0.2557 ± 0.0077	0.9665 ± 0.0024	0.9559 ± 0.0061	0.4559 ± 0.0061	0.9679 ± 0.0029	0.9560 ± 0.0075	0.4560 ± 0.0075	0.9697 ± 0.0033	0.9577 ± 0.0080	0.4577 ± 0.0080

Table 5.11: OOD Metrics for MNIST across various datasets and models. Best results are in **bold**, second-best are underlined. For AUR, the best result is the lowest value.

For MNIST, the trends are similar, though the performance on TinyImageNet and Fashion-MNIST is not as strong.

While the Multi-Output module can slightly improve OOD detection, particularly on certain datasets and head configurations, the improvements are generally modest and not game-changing, the results are better in UQ than OOD detection overall.

5.4 Understanding the method

5.4.1 The training process and the hyperparameters

The module is trained using early stopping, a regularization technique that monitors the loss on the validation set, such as cross-entropy (NLL), during training. The validation set is a subset of the training data, commonly 10-20% of the training set, which is used to tune model parameters and prevent overfitting. It is important to note that the validation set is distinct from the test set, as the test set is reserved for the final evaluation of the model's performance after training is complete, whereas the validation set is used during training to assess model improvements. If the validation loss does not improve for a predefined number of epochs, training is halted to avoid overfitting. This approach helps the model generalize better to unseen data by saving it at the optimal point, balancing model complexity and training time. As described earlier, during inference, we repeat inputs and average the outputs across heads on the validation set. A detailed pseudocode can be found at the end of the thesis: 2. The resulting number of epochs and training times for different datasets are obtained over 5 runs shown on table 5.12.

We can clearly see that training time increases as the number of heads grows, which is expected due to the corresponding rise in the number of parameters. Note that the module has been trained on a single NVIDIA GeForce RTX 2080, this is to be compared to the base WideResNet-28x10 on the cifar10 datasets that required 25h 08m 40s for training on two NVIDIA GeForce RTX 2080.

Since the module is trained by minimizing the sum of losses, which corresponds to maximizing the joint likelihood, it is better able to capture uncertainty compared to an ensemble, particularly in cases of class overlap as shown in 5.4, such as when a 5 resembles a 9.

There are couple standards hyperparameters to consider here, and we got our results with these ones:

Model	**nb epochs**	**training time**	**nb parameters**
CIFAR-10			
Multi-Output 3 heads	36.6000 ± 11.9432	0h 35m 38s ± 0h 11m 19s	36,510,380
Multi-Output 5 heads	61.8000 ± 15.4454	1h 13m 20s ± 0h 17m 57s	36,526,440
Multi-Output 7 heads	75.0000 ± 10.9909	1h 11m 33s ± 0h 10m 22s	36,544,100
Multi-Output 10 heads	90.0000 ± 11.7303	1h 39m 41s ± 0h 13m 20s	36,573,590
CIFAR-100			
Multi-Output 3 heads	20.2000 ± 3.3106	0h 22m 0s ± 0h 3m 20s	36,919,880
Multi-Output 5 heads	29.2000 ± 7.1944	0h 42m 8s ± 0h 10m 34s	37,368,480
Multi-Output 7 heads	46.4000 ± 8.2849	0h 55m 2s ± 0h 9m 56s	37,977,080
Multi-Output 10 heads	60.2000 ± 9.5791	1h 23m 45s ± 0h 11m 13s	39,189,980
MNIST			
Multi-Output 3 heads	93.8000 ± 25.2301	0h 5m 11s ± 0h 1m 22s	48,836
Multi-Output 5 heads	138.0000 ± 35.4570	0h 7m 26s ± 0h 1m 53s	53,776
Multi-Output 7 heads	102.4000 ± 7.4189	0h 5m 44s ± 0h 0m 26s	60,316
Multi-Output 10 heads	159.0000 ± 22.1721	0h 9m 35s ± 0h 1m 17s	73,126
Multi-Output 13 heads	170.0000 ± 33.7402	0h 11m 29s ± 0h 2m 16s	89,536
Multi-Output 15 heads	167.0000 ± 22.7156	0h 12m 5s ± 0h 1m 34s	102,476

Table 5.12: Number of epochs, training time, and number of parameters for each model across CIFAR-10, CIFAR-100, and MNIST.

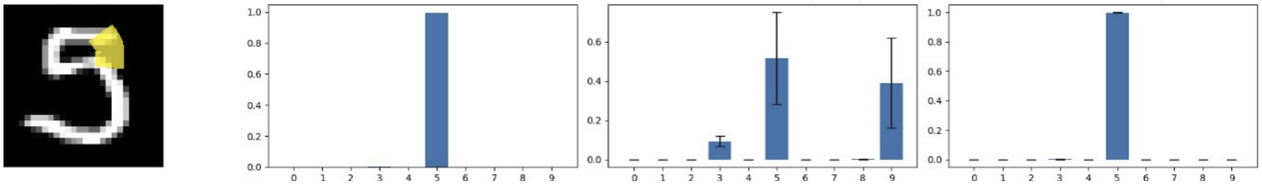


Figure 5.4: Conventional neural networks generate marginal predictions on the left that do not separate joint distribution ambiguity, the module is in the center and the ensemble of base models on the right.

Dataset	**Learning Rate (LR)**	**L2 Weight Decay**	**Batch Size**	**Optimizer**
CIFAR	0.0001	0.0005	16 × num_heads	Adam
MNIST	0.0005	0	16 × num_heads	Adam

Table 5.13: Hyperparameters for CIFAR and MNIST datasets.

5.4.2 Trade-off between accuracy and calibration

In the MNIST and CIFAR-10 settings, we observe the trade-off between accuracy and ECE as the number of heads increases. Particularly as shown in 5.5 and 5.6 using LeNet5 as the base model

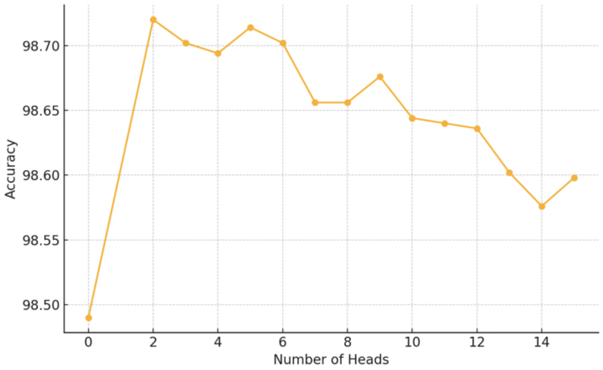


Figure 5.5: Accuracy with the Number of Heads

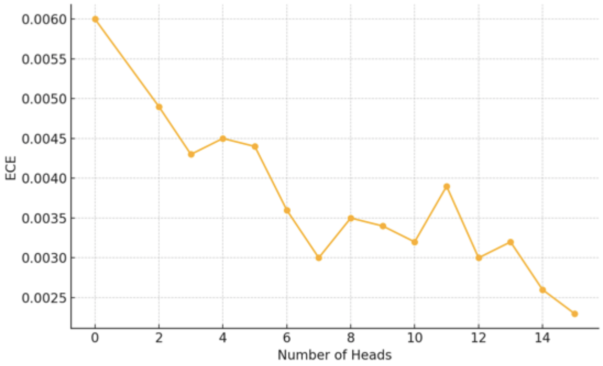


Figure 5.6: ECE with the Number of Heads

Accuracy shows a slight decline after a certain point as the number of heads increases. Starting from the base model's accuracy (98.49%), the best accuracy is around 98.72% with 2 heads, remaining relatively stable until about 5-6 heads, after which it begins to decrease slightly.

In contrast, ECE improves significantly as the number of heads increases, reaching its lowest value around 15 heads, indicating better calibration. The most noticeable improvement occurs between 2 and 7 heads, highlighting the impact of adding heads on calibration. This shows a trade-off between accuracy and calibration when choosing the optimal number of heads.

This trade-off is further illustrated through proper scoring metrics in 5.7 and 5.8.

NLL improves (decreases) with up to 5 heads but worsens slightly beyond 6 heads. The Brier score remains relatively unchanged, showing little variation across different numbers of heads. With 5 heads, we observe the best balance between high accuracy (98.71%) and good calibration (NLL = 0.0404, ECE = 0.0044). This represents the point where calibration benefits from more heads, but accuracy starts to decline. Beyond 5 heads, while calibration continues to improve, accuracy drops more noticeably.

Thus, selecting the number of heads that optimizes a proper scoring rule (NLL or the Brier score) provides an ideal balance between accuracy and calibration.

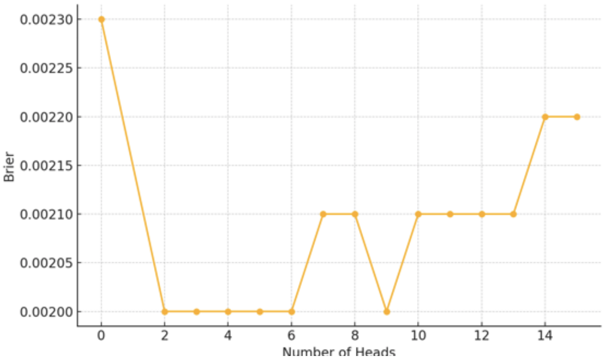


Figure 5.7: Brier Score with the Number of Heads

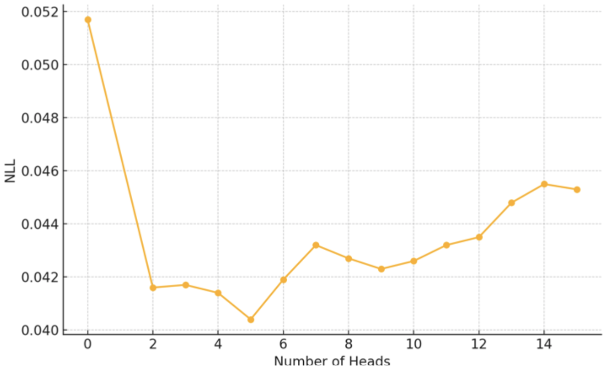


Figure 5.8: NLL with the Number of Heads

Chapter 6

Conclusion

In this study, we introduced a novel post-hoc uncertainty estimation technique that operates as a meta-model on top of a pre-trained model. Our method generates multiple independent predictions in a single forward pass without requiring additional data or retraining, it is both model-agnostic and highly efficient. Despite its simplicity, our approach achieves near state-of-the-art performance on MNIST, CIFAR-10, CIFAR-100, and their corrupted versions, with minimal computational overhead.

What makes our methodology stand out is its ability to deliver robust uncertainty quantification while drastically reducing training time and resource requirements, unlike conventional methods that involve full model retraining and sometimes additional data. By exploiting the disagreement between outputs, we efficiently disentangle uncertainties without compromising accuracy or reliability. This efficiency makes it a highly scalable solution for real-world applications, with potential for extension to regression tasks.

That said, further exploration is needed, particularly in testing this method across more diverse datasets and benchmarking it against other post-hoc techniques, especially in out-of-distribution (OOD) detection.

We would also like to highlight an often-overlooked aspect in uncertainty estimation: numerical uncertainty, particularly due to approximation errors. In high-dimensional optimization problems, it is frequently unclear if the algorithm has fully converged, leading to discrepancies between the true minimum and the optimizer's result. This is especially critical in real-time systems like self-driving cars. Rounding errors can further exacerbate uncertainty, as discussed by Gupta et al. [2015]. Addressing these approximation errors is crucial for improving the robustness of uncertainty estimation methods.

Our method, while promising in its efficiency and scalability, also opens up opportunities for further refinement and broader application.

Bibliography

Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, Dana Rezazadegan, Li Liu, Mohammad Ghavamzadeh, Paul Fieguth, Xiaochun Cao, Abbas Khosravi, U Rajendra Acharya, Vladimir Makarenkov, and Saeid Nahavandi. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *arXiv preprint arXiv:2011.06225*, 2021.

Yong Hyun Ahn, Gyeong-Moon Park, and Seong Tae Kim. LINe: Out-of-distribution detection by leveraging important neurons. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. doi: 10.48550/arXiv.2303.13995.

Alexander Amini, Wilko Schwarting, Ava Soleimany, and Daniela Rus. Deep evidential regression. In *Advances in Neural Information Processing Systems*, volume 33, pages 14927–14937, 2020.

Anastasios N Angelopoulos and Stephen Bates. A gentle introduction to conformal prediction and distribution-free uncertainty quantification. *arXiv preprint arXiv:2107.07511*, 2021.

Edmon Begoli, Tanmoy Bhattacharya, and Dimitri Kusnezov. The need for uncertainty quantification in machine-assisted medical decision making. *Nature Machine Intelligence*, 1(1):20–23, 2019.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35 (8):1798–1828, 2013.

Viktor Bengs, Eyke Hüllermeier, and Willem Waegeman. On second-order scoring rules for epistemic uncertainty quantification. *arXiv preprint arXiv:2301.12736*, 2023.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

George E. P. Box and George C. Tiao. *Bayesian Inference in Statistical Analysis*. John Wiley & Sons, New York, 1973.

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

Bertrand Charpentier, Alexander Amini, Wilko Schwarting, Joschka Boedecker, Pascal Wolfram, and Daniela Rus. Uncertainty quantification for evidential deep learning. *Advances in Neural Information Processing Systems*, 35:20963–20977, 2022.

Ying Chen, Andre Biedenkapp, Jan N. van Rijn, and Frank Hutter. Meta-surrogate benchmarking for hyperparameter optimization. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1039–1049. ACM, 2019.

Muthu Chidambaram and Rong Ge. On the limitations of temperature scaling for distributions with overlaps. *International Conference on Learning Representations (ICLR)*, 2024. URL `https://arxiv.org/abs/2306.00740`. 27 pages, 9 Figures.

Jiwoong Choi, Dayoung Chun, Hyun Kim, and Hyuk-Jae Lee. Gaussian YOLOv3: An accurate and fast object detector using localization uncertainty for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 502–511, 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. Learning and policy search in stochastic dynamical systems with bayesian neural networks. *arXiv preprint arXiv:1605.07127*, 2017. doi: 10.48550/arXiv.1605.07127. URL `https://arxiv.org/abs/1605.07127`. Version 3.

Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Decomposition of uncertainty in bayesian deep learning for efficient and risk-sensitive learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 1192–1201. PMLR, 2018.

Thomas G. Dietterich. Ensemble methods in machine learning. In Josef Kittler and Fabio Roli, editors, *Multiple Classifier Systems*, pages 1–15. Springer, Berlin, Heidelberg, 2000.

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

Ronald A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.

Yarin Gal. *Uncertainty in Deep Learning.* PhD thesis, University of Cambridge, Cambridge, UK, 2016.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, pages 1050–1059. PMLR, 2016.

Tilmann Gneiting and Adrian E Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015. URL `https://arxiv.org/abs/1412.6572`.

Google Research. Uncertainty baselines for cifar. `https://github.com/google/uncertainty-baselines/tree/main`, 2023.

Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems 24*, pages 2348–2356, 2011.

Varun Gulshan, Lily Peng, Marc Coram, Martin C Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, Ramasamy Kim, Rajiv Raman, Philip C Nelson, Jessica L Mega, and Dale R Webster. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *JAMA*, 316(22):2402–2410, 2016. doi: 10.1001/jama.2016.17216.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 1321–1330. PMLR, 2017.

Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. *arXiv preprint arXiv:1502.02551*, 2015. URL `https://doi.org/10.48550/arXiv.1502.02551`.

James Harrison, John Willes, and Jasper Snoek. Variational bayesian last layers: Efficient uncertainty estimation in neural networks. In *International Conference on Learning Representations (ICLR)*, 2024.

W.K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

Marton Havasi, Rodolphe Jenatton, Stanislav Fort, Florian Wenzel, Linh Tran, Joshua V Dillon, and Balaji Lakshminarayanan. Training independent subnetworks for robust prediction. *Advances in neural information processing systems*, 33:7586–7598, 2020.

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 (CVPR)*, pages 770–778, 2016.

Wenchong He and Zhe Jiang. A comprehensive survey on uncertainty quantification for deep learning. *arXiv preprint arXiv:2302.13425*, 2024. License: CC BY-NC-ND 4.0.

Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019. URL `https://doi.org/10.48550/arXiv.1903.12261`. ICLR 2019 camera-ready.

Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2017.

Dan Hendrycks, Norman Mu, Ekin D Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. Augmix: A simple data processing method to improve robustness and uncertainty. In *International Conference on Learning Representations*, 2020.

José Miguel Hernández-Lobato, Yingzhen Li, Mark Rowland, Daniel Hernández-Lobato, Thang Bui, and Richard E. Turner. Black-box α-divergence minimization. *arXiv preprint arXiv:1511.03243*, 2016. URL `https://arxiv.org/abs/1511.03243`. Accepted at ICML 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. In *International Conference on Learning Representations (ICLR)*, 2017.

Varun Jain, Julien Ton, Vincent Fortuin, Piyush Rai, and Andrew Gordon Wilson. Deup: Direct epistemic uncertainty prediction. In *Proceedings of the 38th International Conference on Machine Learning*, pages 4691–4700. PMLR, 2021.

Trevor Hastie Jerome Friedman and Robert Tibshirani. *The Elements of Statistical Learning*. Springer, 2001.

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.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in Neural Information Processing Systems*, pages 5574–5584, 2017.

Armen Der Kiureghian and Ove Ditlevsen. Aleatory or epistemic? does it matter? *Structural Safety*, 31(2):105–112, 2009. Risk Acceptance and Risk Communication.

Anna-Kathrin Kopetzki, Bertrand Charpentier, Daniel Zügner, Sandhya Giri, and Stephan Günnemann. Evaluating robustness of predictive uncertainty estimation: Are dirichlet-based models reliable? *arXiv preprint arXiv:2010.14986*, 2021.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105, Red Hook, NY, USA, 2012. Curran Associates, Inc.

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

Stefan Larson, Anish Mahendran, Joseph J. Peper, Christopher Clarke, Andrew Lee, Parker Hill, Jonathan K. Kummerfeld, Kevin Leach, Michael A. Laurenzano, Lingjia Tang, and Jason Mars. An evaluation dataset for intent classification and out-of-scope prediction.

arXiv preprint arXiv:1909.02027, 2019. URL `https://doi.org/10.48550/arXiv.1909.02027`. Accepted to EMNLP-IJCNLP 2019.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*, 2018.

Jeremiah Zhe Liu, Zi Lin, Shreyas Padhy, Dustin Tran, Tania Bedrax-Weiss, and Balaji Lakshminarayanan. Simple and principled uncertainty estimation with deterministic deep learning via distance awareness. *arXiv preprint arXiv:2006.10108*, 2020a. URL `https://doi.org/10.48550/arXiv.2006.10108`.

Weitang Liu, Xiaoyun Wang, John D Owens, and Yixuan Li. Energy-based out-of-distribution detection. In *Advances in Neural Information Processing Systems*, 2020b.

Wesley Maddox, Andrew Gordon Wilson, and Pavel Izmailov. A simple baseline for bayesian uncertainty in deep learning. In *Advances in Neural Information Processing Systems*, 2019.

Andrey Malinin and Mark Gales. Predictive uncertainty estimation via prior networks. *Advances in neural information processing systems*, 31, 2018.

Charles C. Margossian, Loucas Pillaud-Vivien, and Lawrence K. Saul. Variational inference for uncertainty quantification: an analysis of trade-offs. *arXiv preprint arXiv:2403.13748*, 2024. URL `https://doi.org/10.48550/arXiv.2403.13748`. Last revised on 7 Jun 2024 (v2).

Antonio Ruiz Masegosa. Bayesian deep learning: A review. *arXiv preprint arXiv:2006.06836*, 2020.

Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

Lu Mi, Hao Wang, Yonglong Tian, Hao He, and Nir Shavit. Training-free uncertainty estimation for dense regression: Sensitivity as a surrogate. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence*, 2022. Available at `https://doi.org/10.48550/arXiv.1910.04858`.

Thomas Minka. Divergence measures and message passing. Technical Report MSR-TR-2005-173, Microsoft Research Ltd., Cambridge, UK, December 7 2005.

Devina Mohan, Anna M M Scaife, Fiona Porter, Mike Walmsley, and Micah Bowles. Quantifying uncertainty in deep learning approaches to radio galaxy classification. *Monthly Notices of the Royal Astronomical Society*, 511(3):3722–3740, 2022. doi: 10.1093/mnras/stac223. URL `https://doi.org/10.1093/mnras/stac223`. Published: 29 January 2022.

Radford M. Neal. *Bayesian Learning for Neural Networks*. Springer, 1994.

Radford M. Neal. *Bayesian Learning for Neural Networks*, volume 118. Springer Science & Business Media, 1996.

Radford M. Neal. Mcmc using hamiltonian dynamics. In *Handbook of Markov Chain Monte Carlo*, pages 113–162. CRC Press, 2011.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. *arXiv preprint arXiv:1412.1897*, 2015. URL `https://doi.org/10.48550/arXiv.1412.1897`. To appear at CVPR 2015.

Ian Osband, Zheng Wen, Seyed Mohammad Asghari, Vikranth Dwaracherla, Morteza Ibrahimi, Xiuyuan Lu, and Benjamin Van Roy. Epistemic neural networks. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL `https://proceedings.neurips.cc/paper_files/paper/2023/file/0f42911b2af3e7b2ac820b0ad011f37b-Paper-Conference.pdf`.

Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D Sculley, Sebastian Nowozin, Joshua V. Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. *arXiv preprint arXiv:1906.02530*, 2019. URL `https://doi.org/10.48550/arXiv.1906.02530`. Advances in Neural Information Processing Systems, 2019.

Mahdi Pakdaman Naeini, Gregory Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015. doi: 10.1609/aaai.v29i1.9602. URL `https://doi.org/10.1609/aaai.v29i1.9602`.

Mohit Pandey, Michael Fernandez, Francesco Gentile, Olexandr Isayev, Alexander Tropsha, Abraham C. Stern, and Artem Cherkasov. The transformational role of GPU computing and deep learning in drug discovery. *Nature Machine Intelligence*, 4(3):211–221, 2022.

Barnabás Póczos and Jeff Schneider. On the estimation of α-divergences. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, 2011.

Christian P. Robert and George Casella. *Monte Carlo Statistical Methods*. Springer, 1999.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

Glenn Shafer and Vladimir Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9(Mar):371–421, 2008.

Maohao Shen, Yuheng Bu, Prasanna Sattigeri, Soumya Ghosh, Subhro Das, and Gregory Wornell. Post-hoc uncertainty learning using a dirichlet meta-model. *arXiv preprint arXiv:2212.07359*, 2022. URL `https://doi.org/10.48550/arXiv.2212.07359`. Accepted by AAAI 2023.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015, Conference Track Proceedings*, 2015.

Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Sheng Zhao, Shuyang Cheng, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in perception for autonomous driving: Waymo open dataset. *arXiv preprint arXiv:1912.04838*, 2020. URL `https://doi.org/10.48550/arXiv.1912.04838`. CVPR 2020.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

Linh Tran, Bastiaan S Veeling, Kevin Roth, Jakub Swiatkowski, Joshua V Dillon, Jasper Snoek, Stephan Mandt, Tim Salimans, Sebastian Nowozin, and Rodolphe Jenatton. Hydra: Preserving ensemble diversity for model distillation. *arXiv preprint arXiv:2001.04694*, 2020.

Robert Tibshirani Trevor Hastie and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

Dennis Ulmer, Christian Hardmeier, and Jes Frellsen. Prior and posterior networks: A survey on evidential deep learning methods for uncertainty estimation. *arXiv preprint arXiv:2110.03051*, 2021. URL `https://doi.org/10.48550/arXiv.2110.03051`.

Matias Valdenegro-Toro and Daniel Saromo Mori. A deeper look into aleatoric and epistemic uncertainty disentanglement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, IEEE, 2022.

Joost Van Amersfoort, Lewis Smith, Yee Whye Teh, and Yarin Gal. Simple and scalable predictive uncertainty estimation using deep ensembles and nearest neighbors. In *Advances in Neural Information Processing Systems*, volume 33, pages 14186–14197, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6000–6010. Curran Associates Inc., 2017.

Yeming Wen, Paul Vicol, Jimmy Ba, Dustin Tran, and Roger Grosse. Batchensemble: Efficient ensemble learning by iterative refinement. In *International Conference on Learning Representations (ICLR)*, 2020. URL `https://openreview.net/forum?id=SkgpyxHYwH`.

Florian Wenzel, Kevin Roth, Bastiaan S Veeling, Jakub Swiatkowski, Linh Tran, Stephan Mandt, Jasper Snoek, Tim Salimans, Rodolphe Jenatton, and Sebastian Nowozin. How good is the bayes posterior in deep neural networks really? In *International Conference on Machine Learning (ICML)*, pages 10248–10259. PMLR, 2020.

Andrew Gordon Wilson and Pavel Izmailov. Bayesian deep learning and stochastic gradient descent. *arXiv preprint arXiv:2002.09620*, 2020.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference (BMVC)*, 2016.

Chunyuan Zhang, Michael Rosenstein, Alexander Smola, Xiaodong He, and Eric P Xing. Noisy natural gradient as variational inference. In *International Conference on Machine Learning (ICML)*, pages 5852–5861, 2018.

Xiaojin Zhu. Semi-supervised learning literature survey. *Computer Sciences Technical Report, University of Wisconsin-Madison*, 2005.

Pseudocode

Algorithm 1 Multi-Output Module

```
procedure MultiOutputModule($num\_heads$, $base\_model$, $device$)
    Initialize $num\_heads$, $base\_model$, and $device$.
    Register hook to capture penultimate output from $base\_model$.
    Initialize $input\_heads$ for $num\_heads$ using copied last layer.
    Initialize shared layer and $output\_layers$ for $num\_heads$.
end procedure
procedure forward($x$, $train\_infer$)
    Set $base\_model$ to eval mode, disable gradient calculation.
    Move $x$ to $device$.
    if $train\_infer$ = “training″ then
        Pass $x$ through $base\_model$.
        Capture and apply activation to penultimate layer output.
        Reshape penultimate output for $num\_heads$.
    else if $train\_infer$ = “inference″ then
        Pass $x$ through $base\_model$ once to get final output.
        Capture and apply activation to penultimate layer output.
        Rpeat penultimate output for $num\_heads$.
    end if
    for $i = 1$ to $num\_heads$ do
        Process $i^{th}$ head using its respective input head.
    end for
    Concatenate processed heads and pass through shared layers.
    for $i = 1$ to $num\_heads$ do
        Compute final output for $i^{th}$ head.
    end for
    if $train\_infer$ = ”training” then
        Concatenate all output heads for $num\_heads$ predictions.
    else if $train\_infer$ = ”inference” then
        Append base model’s final output to the predictions.
        Concatenate for $num\_heads + 1$ predictions.
    end if
    Return final predictions.
end procedure
```

Algorithm 2 Training Procedure for Multi-Output Module

```
procedure TrainModule($module$, $train\_loader$, $val\_loader$, $num\_epochs$, $optimizer$, $criterion$, $device$, $patience$)
    Initialize $best\_loss \leftarrow \infty$, $epochs\_no\_improve \leftarrow 0$
    for $epoch = 1$ to $num\_epochs$ do
        $module.train()$
        for all $(images, labels)$ in $train\_loader$ do
            Move $images$, $labels$ to $device$
            $predictions \leftarrow module(images, \text{“}training''\text{)}$
            Compute $total\_loss$ across all heads by summing all the losses
            Backpropagation and update: $optimizer.step()$
        end for
        $module.eval()$
        Evaluate on validation set, compute $val\_loss$ and accuracy
        if $val\_loss < best\_loss$ then
            Save best model state
            $epochs\_no\_improve \leftarrow 0$
        else
            $epochs\_no\_improve \leftarrow epochs\_no\_improve + 1$
        end if
        if $epochs\_no\_improve \geq patience$ then
            Early stopping
            break
        end if
    end for
    Load best model state
end procedure
```