

Claremont Colleges

Scholarship @ Claremont

CGU Theses & Dissertations

CGU Student Scholarship

Fall 2021

Measuring Machine Learning Model Uncertainty with Applications to Aerial Segmentation

Kevin James Cotton

Claremont Graduate University

Follow this and additional works at: https://scholarship.claremont.edu/cgu_etd



Part of the [Applied Mathematics Commons](#), and the [Mathematics Commons](#)

Recommended Citation

Cotton, Kevin James. (2021). *Measuring Machine Learning Model Uncertainty with Applications to Aerial Segmentation*. CGU Theses & Dissertations, 271. https://scholarship.claremont.edu/cgu_etd/271.

This Open Access Dissertation is brought to you for free and open access by the CGU Student Scholarship at Scholarship @ Claremont. It has been accepted for inclusion in CGU Theses & Dissertations by an authorized administrator of Scholarship @ Claremont. For more information, please contact scholarship@cuc.claremont.edu.

Measuring Machine Learning Model Uncertainty with Applications to Aerial Segmentation

By
Kevin J Cotton

Claremont Graduate University

2021

© Copyright Kevin J Cotton, 2021

All rights reserved.

APPROVAL OF THE DISSERTATION COMMITTEE

This dissertation has been duly read, reviewed, and critiqued by the Committee listed below, which hereby approves the manuscript of Kevin J Cotton as fulfilling the scope and quality requirements for meriting the degree of Doctor of Philosophy in Mathematics.

Dr. Allon Percus, Chair
Professor of Mathematics
Claremont Graduate University

Dr. Marina Chugunova
Professor of Mathematics
Claremont Graduate University

Dr. Weiqing Gu
Professor of Mathematics
Harvey Mudd College

ABSTRACT

Measuring Machine Learning Model Uncertainty with Applications to Aerial Segmentation

By

Kevin J Cotton

Claremont Graduate University: 2021

Machine learning model performance on both validation data and new data can be better measured and understood by leveraging uncertainty metrics at the time of prediction. These metrics can improve the model training process by indicating which training data need to be corrected and what part of the domain needs further annotation. The methods described have yet to reach mainstream adoption, and show great potential. Here, we survey the field of uncertainty metrics and provide a robust framework for its application to aerial segmentation.

Uncertainty is divided into two types: aleatoric and epistemic. Aleatoric uncertainty arises from variations in training data and can be the result of poor training data or an inherently stochastic observation. Epistemic uncertainty arises from predicting on inputs that are out of class of the training data. Both measures inform the machine learning engineer on what areas of data need better or more training, and also help downstream processes quantify the usefulness of a prediction.

We survey the current tools for measuring uncertainty, including the autoencoder for measuring epistemic uncertainty and the Bayesian neural network. The latter replaces each trained weight with a random variable with which we approximate the true, unknown distribution of each weight with a two parameter (mean and variance) normal distribution. Bayes by Backprop trains these parameters by minimizing the Kullback–Leibler divergence between the approximating normal distribution and the unknown distribution. Our contribution is a novel application of the Bayesian neural network with Gaussian weights applied to the U-Net model for aerial segmentation.

Using the DroneDeploy dataset, we build and train our Bayesian U-Net model and gather epistemic and aleatoric uncertainty metrics. Experimentally, we find that these metrics are correlated to

model performance on unseen data and thus provide immediate value to a modeling and prediction workflow. We show the usefulness of these metrics for both per-pixel uncertainty estimation and per-image uncertainty estimation.

Acknowledgements

I would like to thank my advisor, Prof. Percus, for his continual support and patience while I refined my topic. I would also like to thank my other committee members, Prof. Chugunova and Prof. Gu, each having spent a good deal of time working with me on my topic. All of whom were instrumental at my time at CGU.

I want to also thank those whom I worked with at JPL during my internship for all the wisdom they imparted and for instilling a passion for research; namely my mentors Daniel Nunes and Karl Mitchell.

I also want to acknowledge my company, Woven Planet, for providing a context to implement the topics I research here.

Finally, I would like to thank my parents and my family for supporting me and my pursuits. I could not do it without them.

Contents

1	What is Uncertainty	1
1.1	Types of Uncertainty	2
1.2	Methods	3
1.3	Roadmap	4
2	Autoencoders	5
2.1	Types of Autoencoders	7
2.2	Measuring Epistemic Uncertainty with an Autoencoder	7
2.3	Variational Autoencoders	9
3	The Bayesian Neural Network	11
3.1	BNN Summary	11
3.2	Construction	12
3.3	Bayes by Backprop	14
3.4	The Loss Function	15
3.5	Connection to VAEs	17
4	Implementations	20
4.1	Analytical Approach	20
4.2	Regression Implementation	21
4.3	MNIST implementation	24
4.4	Characterizing Datasets	27

5	Measuring Uncertainty	33
5.1	Epistemic Uncertainty	33
5.2	Homoscedastic Aleatoric Uncertainty	34
5.3	Heteroscedastic Aleatoric Uncertainty	34
6	Applications to Aerial Segmentation	37
6.1	The DroneDeploy Dataset	38
6.2	U-Net Architecture	40
6.3	Bayesian Aerial Segmentation: A Bayesian U-Net	42
6.4	Related Work	43
7	Results on Aerial Segmentation	45
7.1	Initial Tests	45
7.2	Full Predictions	52
7.3	Varying Data Size	57
8	Conclusions and Further Perspectives	59
8.1	Summary	59
8.2	A Workflow for Improving Model Performance	60
A	Code Excerpts	62
A.1	U-Net Pytorch Code Excerpts	62
A.2	Bayesian U-Net Pytorch Code Excerpts	63

List of Figures

2.1	A generic autoencoder network with encoder and decoder portion.	5
2.2	Reconstruction MNIST data on MNIST trained autoencoder.	6
2.3	Reconstruction of non-MNIST data on MNIST trained autoencoder.	6
2.4	Sample of MNIST data.	7
2.5	Sample of EMNIST data.	8
2.6	Sample of FMNIST data.	8
2.7	Autoencoder reconstruction error histogram on MNIST and non-MNIST data. . . .	9
2.8	Variational Autoencoder (VAE): training vs generative.	10
3.1	Bayesian neural network structure.	11
3.2	VAE connection to BNN.	18
4.1	Regression data generation.	22
4.2	BNN confidence interval on regression problem.	23
4.3	BNN epistemic uncertainty on MNIST and non-MNIST data.	25
4.4	Mean per-pixel values of MNIST data.	28
4.5	Per-pixel variance of MNIST data.	28
4.6	100 random samples from the multivariate normal distribution that characterizes the MNIST dataset.	29
4.7	Flower data set.	30
4.8	Mean values for the flower dataset.	31
4.9	Variance matrix for flower dataset.	31
4.10	Random samples from the flower multivariate random normal distribution.	32

5.1	Plot of data, prediction, and epistemic uncertainty on regression problem.	34
6.1	DroneDeploy dataset example.	38
6.2	U-Net architecture.	41
6.3	U-Net training curve on DroneDeploy dataset.	42
7.1	Bayesian U-Net prediction on disk with high uncertainty.	46
7.2	Bayesian U-Net prediction on ambiguous clutter.	47
7.3	Bayesian U-Net prediction on unambiguous ground.	48
7.4	Bayesian U-Net prediction on washed out building.	49
7.5	Bayesian U-Net prediction on snow.	50
7.6	Bayesian U-Net prediction on ambiguous vegetation.	51
7.7	Correlation between Bayesian U-Net epistemic uncertainty and mIoU.	53
7.8	Bayesian U-Net validation on ground.	54
7.9	Bayesian U-Net validation on snow.	55
7.10	Bayesian U-Net validation on truck.	56
7.11	Bayesian U-Net validation on ground/vegetation.	57
8.1	Proposed workflow for continuous model performance improvement.	60

List of Tables

6.1	Class distribution of DroneDeploy dataset.	40
7.1	Comparison of epistemic and aleatoric uncertainty for varying dataset size.	58

Chapter 1

What is Uncertainty

One can think of model uncertainty as an understanding of what a model does and does not know. [18] Specifically, machine learning model **uncertainty** can be defined as a metric or set of metrics that give indication of the confidence of the model prediction on any given input data. There are different quantitative metrics of uncertainty, but we can broadly define such metrics as the probability that a given prediction is correct. Some such metrics can be conveniently mapped to $[0,1]$, with 0 being the least likely to be an accurate prediction, and 1 being the most likely. [32]

A neural network has a set of weight that are set at training through backpropagation. For classification tasks, many networks end with a softmax function. The softmax function maps all real values to $[0,1]$, providing a probabilistic interpretation. For example, 1 is a high confidence prediction while 0.01 is a very low confidence prediction.

The problem that an uncertainty metric is trying to solve can easily be seen when feeding out-of-class data to a classification neural network. The classification model may naively predict the closest in-class label with a high softmax output. [16] The softmax output of a classification network that is thresholded and used to assign the class can be a poor indicator of model certainty and may fail to flag data outside the manifold of the training data. This is an example of epistemic uncertainty. For instance, an image classification model that was never trained on samples of dogs will have no ability to predict “dog” when fed such an image. The challenge is to enable the model to report that it does not know what the class is, rather than making some other arbitrary prediction.

A different kind of uncertainty arises from noise in the training data. Aleatoric uncertainty can be a result of mislabeled classes or targets in the training set, or of a natural stochasticity of the targets. A simple example is if some cats were mislabeled as dog in the training set. Another example, in the case of a regression model with a real number target, would be a certain area of the domain that has training samples giving a large degree of variance in the output.

In all of these cases, we seek to build a model that gives an associated certainty metric for each output. When a model makes a prediction, we would like to know how certain it is of its prediction.

1.1 Types of Uncertainty

Ståhl et al. [32] and Kana [17] review the two types of uncertainty that we mention above.

- **Epistemic Uncertainty:** uncertainty that is derived from what the model does not know. The input is outside of the manifold of training values. This uncertainty can be reduced by including more training values.
- **Aleatoric Uncertainty:** uncertainty that arises from stochasticity of the observations. This level of uncertainty cannot be reduced by including more training data, but could possibly be reduced by more consistent training data. (Unless the stochasticity is inherent in the data.) Aleatoric uncertainty is localized.

These types of uncertainty will be further discussed in subsequent chapters. Note that, when considering model accuracy on new data, one must also consider the model chosen and its ability to correctly capture the structure of the data. Though we will not cover it here, this kind of uncertainty that arises from the model choice and its parameters can be defined as **Structural Uncertainty**.

All of these uncertainties are inherently hard to measure. When feeding an unknown class to EfficientNet, it can predict the wrong class with very high (>80%) certainty. In their example, Hüllermeier et al. [16] show how EfficientNet will confidently predict panda cookies as a “typewriter” and a pile of logs as a “stone wall.”

This measure of certainty, as we can see, is very poor for these examples. This is an example

of using the last layer of the network, the softmax layer, as a fuzzy indicator of certainty. Since we assign the class as that which corresponds to the node with the highest softmax output, it would make sense to use this output as a proxy for certainty. But, as we can see and as shown experimentally, that is quite a poor metric when the input class was not in the domain of training data.

1.2 Methods

Finding a robust metric for uncertainty is an open problem in Machine Learning. There exists a few methods that Ståhl et al. explore [32], with much room for research still open. I have adapted this list, and added to it with my own methods that we will explore in more detail.

- Using the **Softmax Output** of a deep neural network as a proxy for certainty. This is the least robust method, and has been shown to give unreliable results. The softmax output is rightly thresholded and used to assign a prediction, but it does not give a good measure of model certainty. Further, this method does not work for a regression output.
- Use of an **Autoencoder Network** as a separate network on the training set gives a different proxy metric of certainty. The autoencoder can signal inputs that lie outside the training domain, and thus will have a lower certainty. Autoencoders are of interest in their own right, in that they act unsupervised on a training data set. They can be thought of as a clustering technique. Any new input then has a metric of how close or far it is from the training dataset.
- The **Bayesian Neural Network**, or **BNN**, is our main subject of exploration and contribution. Through training random weights, we produce a random model that gives a stochastic output. This property is leveraged to produce not only a prediction, but also metrics for epistemic and aleatoric uncertainty.
- Characterizing data as a **multivariate random normal distribution**: This method acts as a simplified variational autoencoder and provides a metric for any piece of data to the dataset characterization. We can use this metric as a proxy for uncertainty.

1.3 Roadmap

In this work, we will first explore types of uncertainty arising in machine learning. We distinguish between aleatoric and epistemic uncertainties, and then further divide aleatoric uncertainty into homoscedastic and heteroscedastic varieties. We survey the current methods for producing an uncertainty estimate, including: using the softmax output, the autoencoder, and the Bayesian neural network.

In Chapter 2, we explore the autoencoder in more depth, and then train on MNIST data. We then show that, when fed new, out of class data, the trained autoencoder gives a higher metric of uncertainty, serving as a good proxy for aleatoric uncertainty.

Chapter 3 introduces the Bayesian neural network (BNN): a network that replaces the trained weights with trained probability distributions. We go through the theory of the BNN, and how one trains and predicts with it.

Chapter 4 discusses our implementation of methods to characterize uncertainty. Section 4.1 presents an analytical framework for a fully connected BNN. We apply a BNN implementation by Blundell, et al. [4] to a polynomial regression problem. Section 4.3 applies the BNN to MNIST data and explores the calculated uncertainties. Finally, Section 4.4 considers a simplified method of characterizing a dataset that aims to achieve similar results to the autoencoder.

Chapter 5 continues with the regression implementation of the BNN and further studies ways of measuring uncertainty. Our main contribution is in chapter 6, where we turn our attention to aerial segmentation. We develop a novel Bayesian U-Net model with Gaussian weights, and train on the DroneDeploy dataset. We show that the Bayesian U-Net can effectively be used to produce both per-pixel and image wide metrics for both epistemic and aleatoric uncertainty. We demonstrate this experimentally, with examples of each metric.

We conclude by offering some further perspectives on the application of uncertainty metrics to a comprehensive workflow for improving machine learning model performance.

Chapter 2

Autoencoders

The autoencoder is a useful tool for measuring epistemic uncertainty; that is, it does well at characterizing the manifold of training data and giving a metric of closeness to this manifold for any new input data. When the Euclidean distance metric from the autoencoder is high, then the expected aleatoric certainty from the model is low. A schematic for an autoencoder is given in Figure 2.1.

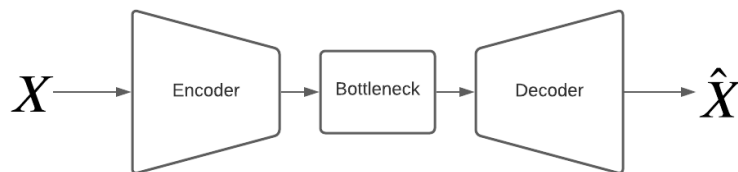


Figure 2.1: A generic autoencoder network with encoder and decoder portion. X is the input image, and \hat{X} is the reconstructed output image.

Grover et al. show how we can use autoencoders to learn compressed projections of data. [13] Essentially, they reduce the dimensionality of an input data through downsampling into a bottleneck, and then upsample to the original. The bottleneck provides a limit for how much information is passed through; and the network is trained to minimize the RMSE (or other kind of error) between

the input and the output data. RMSE is defined as

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (\hat{x}_i - x_i)^2} \quad (2.1)$$

Where the sum is over all n pixels, x_i is the original pixel value, and \hat{x}_i is the reconstructed pixel value. This same metric is used on new input data and its re-projection at prediction time to give a measure of closeness to the training data. When this metric is high, and above a certain, set threshold, we can assume the the data is different enough from the training data that any model from the training data will not give a good prediction.

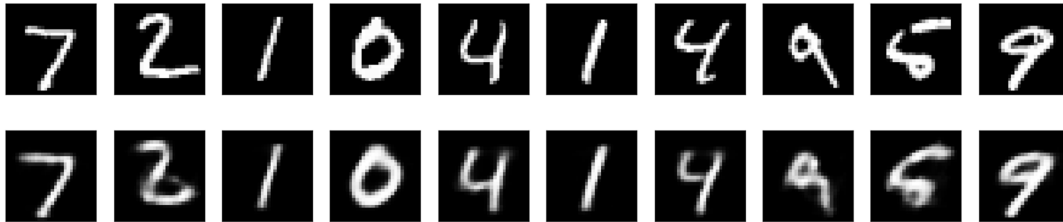


Figure 2.2: We have trained an autoencoder on MNIST data and show MNIST reconstructions here. See section 2.2 for more on the MNIST dataset.



Figure 2.3: Using the same model as above, we look at the reconstruction of out-of-domain data on the MNIST autoencoder. The reconstructions do not resemble the inputs well in this case.

We illustrate this point in figure 2.2 and 2.3 above. A single autoencoder is trained on MNIST data. When reconstructing MNIST data, the reconstruction error is low; but, when reconstructing non-MNIST data (out of class data), the reconstruction error is high. The encoding has learned many features about MNIST data, but does not know as much about non-MNIST data.

2.1 Types of Autoencoders

An autoencoder can take the form of a fully connected network, a convolutional network, or others. For convolutional autoencoders, max pooling is used to downsample the image and up-sampling is used to reconstruct it. Ståhl et al. [32] use a fully connected autoencoder with the following number of neurons, in order: 1000, 250, 50, 10, 50, 250, 1000. The reduction in neurons essentially downsamples the data, then it is upsampled with an increase in neurons.

A convolutional autoencoder downsamples with pairs of convolution followed by max pooling, and upsamples with transpose convolution layers. Normal convolution is either shape preserving (with unit stride), or a downsampling operation (with a stride greater than 1). Transpose convolution, on the other hand, increases the dimensionality by using a fractional stride. [10] It is a very useful upsampling operation that is employed in segmentation tasks. [35] Another kind of upsampling that has more recently been applied to image segmentation is atrous convolution, as described in the groundbreaking paper and architecture *DeepLabv3* [6] The atrous convolution layer has very recently been applied to the autoencoder, and has been coined an atrous convolution autoencoder (A-CAE). [36]

2.2 Measuring Epistemic Uncertainty with an Autoencoder

We will demonstrate how one can train an autoencoder to capture a characterization of a dataset, and then use it measure epistemic uncertainty on new data.

For this example we are using the standard MNIST dataset of hand drawn numbers from 0-9. [21] This is a dataset of 60,000 training and 10,000 test data, each a grayscale image of shape 32x32. Figure 2.4 shows a small random sample.



Figure 2.4: Sample of 10 images from the regular MNIST dataset. Each data is a 32x32 grayscale picture of a hand drawn digit from 0-9.

This will serve as our in-class dataset, and what we train our model on. The autoencoder will

be taught to learn a representation of the MNIST data through the process of compressing and decompressing. When data that is not MNIST (non-MNIST) is fed into the model, it will have a higher reconstruction error.

As our non-MNIST data, we will use the Extended MNIST (EMNIST) dataset, shown in Figure 2.5, that adds hand drawn lowercase letters from a-z and uppercase letters from A-Z. [7] We will also use the Fashion MNIST (FMNIST) dataset, shown in Figure 2.6, that is exclusively grayscale images of articles of clothing. [34] All of these datasets have the same data shape and are grayscale.



Figure 2.5: Sample of 10 images from the regular Extended MNIST (EMNIST) dataset.



Figure 2.6: Sample of 10 images from the regular Fashion MNIST (FMNIST) dataset.

We train a model on just the MNIST data in order to learn a representation of such data. We have split the data into a training and test set. The test set of MNIST, along with non-MNIST data is then passed through the trained autoencoder model to predict certainty. As is shown in Figure 2.7, the reconstruction error is, on average, lower for unseen MNIST data than for non-MNIST data.

To use this model in practice to gauge whether data is in-class or out-of-class requires selection of a proper threshold, and thus a tolerance for higher alpha or beta errors.

Predicting on out-of-class data is an example of epistemic uncertainty. The model has no categories for letters, only digits. Thus, this is only part of the puzzle. To understand aleatoric uncertainty, we must pair the data with a prediction model and look at prediction accuracy.

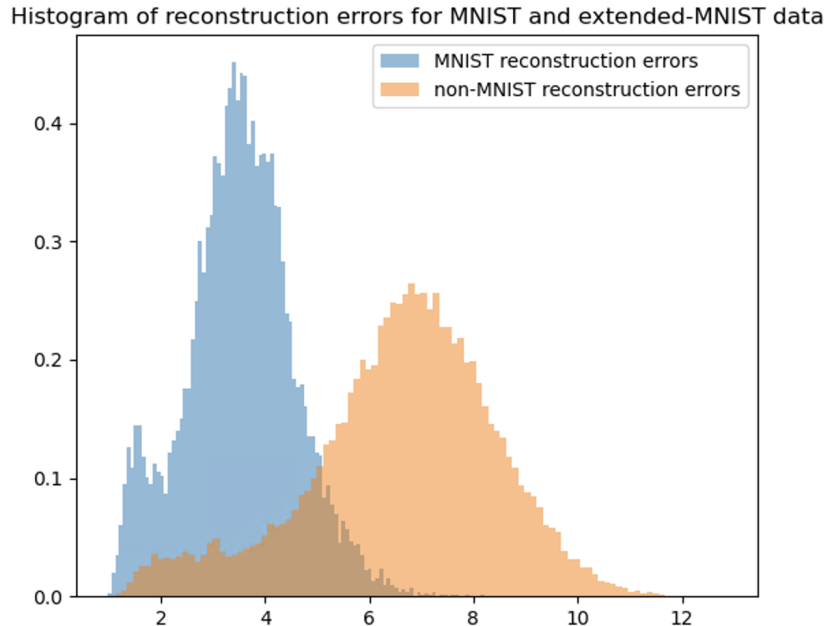


Figure 2.7: Histogram of autoencoder reconstruction error of MNIST and extended-MNIST (characters only) data. X-axis is RMSE of the reconstruction, where units are in pixel values from 0-255. Y-axis is normalized frequency.

2.3 Variational Autoencoders

Variational autoencoders (VAEs) are a type of autoencoder with a Bayesian bottleneck that serve as a regularized representation of the dataset in compressed, latent space. Accordingly, one can sample from the trained latent space, passing the sample through the decoder to re-construct an output, and use the VAE in a generative fashion. Traditional autoencoders do not serve well for this purpose, as the meaningful latent space has very arbitrary values, and it is not known which values to use for generative tasks. [24]

Rocca provides an introduction to the mechanics of VAEs. [26] In figure 2.8, we see the difference between the training and generative process of the VAE. In training, input data is passed through an encoder and downsampled to a latent space representation before it is decoded to re-construct the input. The reconstruction error plus the KL divergence (acting as a regularizer) is then backpropagated to update the layer weights. In the generative process, one samples from the prior distribution on the latent space, and then passes the sample through the decoder to get a generative output.

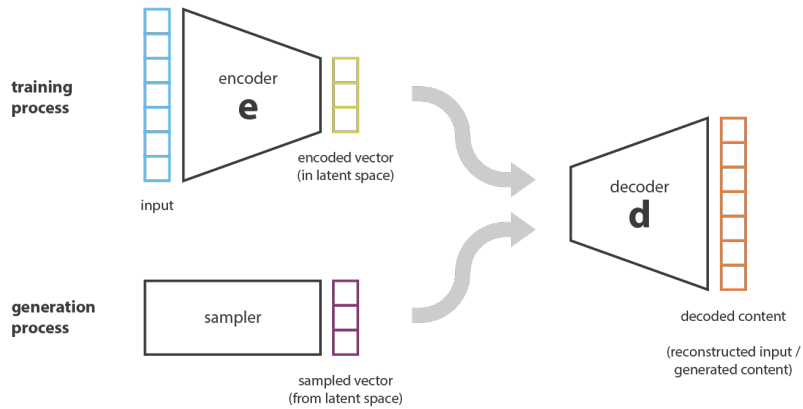


Figure 2.8: Detail from Rocca on how a VAE can be used in a generative fashion, by sampling from the latent space and passing this sample through the decoder. Figure reproduced from [26].

As we will see in section 3.5, VAEs have a strong connection to BNNs and can be considered a subset of BNNs as the bottleneck is similarly trained by minimizing the KL divergence between a Gaussian and an unknown posterior. There, we will explain this connection further after having looked at the mechanics of the BNN.

Chapter 3

The Bayesian Neural Network

3.1 BNN Summary

In their paper, Blundell et al. introduce “a new, efficient, [and] principled” algorithm for constructing and backpropagating on a Bayesian neural network. [4] A Bayesian neural network (or BNN) is a neural network (NN) in which the weights have been replaced with random variables. In the process of training, we do not change the weights directly, but instead train the parameters of each weight. In this chapter, we will look more closely at the implementation by Blundell et al., and construct our own simple BNN on a regression task.

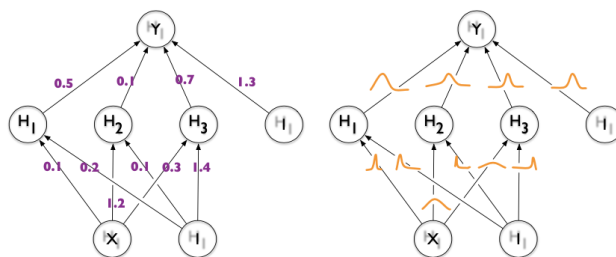


Figure 3.1: Here, we reproduce the figure from Blundell et al. [4]. On the left is traditional neural network with fixed weights that are set at training. On the right, the weights are replaced by distributions whose parameters are set during training.

The motivation of using a BNN as opposed to a traditional NN where the weights are trained and fixed is that BNNs lend well to producing uncertainty metrics. Building off of work by Kendall

and Gal [18] and Kwon et al. [20], we will first implement a data dependent metric on epistemic uncertainty, and then construct a metric for aleatoric uncertainty.

There are two types of metrics for aleatoric uncertainty. Homoscedastic uncertainty is constant for any input after training, and measures the overall aleatoric noise in the model. This metric gives a scalar that can be loosely interpreted as noise of training data overall. Heteroscedastic uncertainty, on the other hand, is data dependent and seeks to give a localized metric of aleatoric noise. A good heteroscedastic uncertainty metric will be very helpful in identifying where noise exists in the training data.

3.2 Construction

We can consider a traditional NN as a probabilistic model $P(y|x, \mathbf{w})$, where for a given input $x \in \mathbb{R}^p$ and trained set of weights \mathbf{w} , a prediction $y \in \mathcal{Y}$ is given. In a Bayesian neural network (BNN), \mathbf{w} are independent random variables whose parameters have been trained by a technique called Bayes by Backprop [4].

Further, for a regression model, output $y \in \mathcal{Y}$ provides a real number \mathbb{R} and $P(y|x, \mathbf{w})$ is defined to be a Gaussian distribution (as we will see, this corresponds to the squared loss).

For a traditional NN, we can train as follows. For a set of N training data $\mathcal{D} = \{(x_i, y_i) | i \in [1, \dots, N]\}$, we seek to find \mathbf{w} that maximize $P(\mathcal{D}|\mathbf{w})$. Using maximum likelihood estimation (MLE), we seek to find, through gradient descent:

$$W^{MLE} = \arg \max_{\mathbf{w}} \log P(\mathcal{D}|\mathbf{w}) \tag{3.1}$$

$$= \arg \max_{\mathbf{w}} \log \prod P(y_i|x_i, \mathbf{w}) \tag{3.2}$$

$$= \arg \max_{\mathbf{w}} \sum \log P(y_i|x_i, \mathbf{w}) \tag{3.3}$$

For a BNN, on the other hand, we seek to find the posterior distribution $P(\mathbf{w}|\mathcal{D})$, given a prior on

the weights $P(\mathbf{w})$. Using Bayes formula, we have:

$$P(\mathbf{w}|\mathcal{D}) = \frac{P(\mathcal{D}|\mathbf{w})P(\mathbf{w})}{P(\mathcal{D})} \quad (3.4)$$

It is not enough to say that the posterior is proportional to the normalized product of the likelihood and prior, for we would like a complete description of the posterior that we can sample from directly. More specifically, as we see above, the posterior $P(\mathcal{D}|\mathbf{w})$ cannot be computed directly. Thus, the true posterior distribution $P(\mathbf{w}|\mathcal{D})$ is intractable, but can be estimated by a variational approach.

Graves [12] and previously Hinton and Van Camp [15] proposed using a variational method to find parameters θ of q that minimizes the Kullback-Leibler (KL) divergence between the true posterior $P(\mathbf{w}|\mathcal{D})$ and approximating posterior $q(\mathbf{w}|\theta)$. The KL divergence is derived from Shannon's source coding theorem [12,29], and though not symmetric, provides a good approximation of twice the squared Fisher distance. [3]

The KL divergence is one of a collection of metrics that measure the distance between two probability distributions that can be considered. Others include the Kolmogorov-Smirnov distance, total variation, and the Kantorovich-Wasserstein metric, which is defined using the optimal transport problem [2]. The KL divergence, in addition to approximating the squared Fisher metric, has some useful properties we will consider.

We let q be a Gaussian distribution with parameters $\theta = (\mu, \sigma)$. The KL divergence is defined as follows:

$$\text{KL}[q(\mathbf{w}|\theta)||P(\mathbf{w}|\mathcal{D})] = \int q(\mathbf{w}|\theta) \log \frac{q(\mathbf{w}|\theta)}{P(\mathcal{D}|\mathbf{w})P(\mathbf{w})} d\mathbf{w} \quad (3.5)$$

We then want to find θ that minimizes the KL divergence, and define our cost function as:

$$F(\mathcal{D}, \theta) = \text{KL}[q(\mathbf{w}|\theta)||P(\mathbf{w}|\mathcal{D})] \quad (3.6)$$

$$= \int q(\mathbf{w}|\theta) \log \frac{q(\mathbf{w}|\theta)}{P(\mathcal{D}|\mathbf{w})P(\mathbf{w})} d\mathbf{w} \quad (3.7)$$

$$= \int q(\mathbf{w}|\theta) (\log q(\mathbf{w}|\theta) - \log P(\mathcal{D}|\mathbf{w}) - \log P(\mathbf{w})) d\mathbf{w} \quad (3.8)$$

$$= \mathbb{E}_{q(\mathbf{w}|\theta)} [\log q(\mathbf{w}|\theta) - \log P(\mathcal{D}|\mathbf{w}) - \log P(\mathbf{w})] \quad (3.9)$$

To do this, we use the gradient descent update algorithm that is simplified as:

$$\theta \leftarrow \theta - \alpha \frac{\partial F}{\partial \theta} \quad (3.10)$$

3.3 Bayes by Backprop

Blundell et al. [4] have shown that, in our case, the partial derivative of the expectation can be re-written as follows. Here ϵ is a random variable with density $q(\epsilon)$ and $\mathbf{w} = t(\theta, \epsilon)$ and t is a deterministic function:

$$\frac{\partial}{\partial \theta} \mathbb{E}_{q(\mathbf{w}|\theta)} [f(\mathbf{w}, \theta)] = \mathbb{E}_{q(\epsilon)} \left[\frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} \frac{\partial \mathbf{w}}{\partial \theta} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \theta} \right] \quad (3.11)$$

We let $f(\mathbf{w}, \theta) = \log q(\mathbf{w}|\theta) - \log P(\mathcal{D}|\mathbf{w}) - \log P(\mathbf{w})$, and seek to find $\frac{\partial}{\partial \theta} \mathbb{E}_{q(\mathbf{w}|\theta)} [f(\mathbf{w}, \theta)]$.

So that the standard deviation would be well behaved while training (σ should remain positive), let us parameterize σ as $\sigma = \log(1 + \exp(\rho))$. Then,

$$\frac{\partial F}{\partial \mu} = \mathbb{E}_{q(\epsilon)} \left[\frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} \frac{\partial \mathbf{w}}{\partial \mu} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \mu} \right] \quad (3.12)$$

$$\frac{\partial F}{\partial \rho} = \mathbb{E}_{q(\epsilon)} \left[\frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} \frac{\partial \mathbf{w}}{\partial \rho} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \rho} \right] \quad (3.13)$$

In order to evaluate the expectations we approximate F using Monte Carlo sampling. Since $\mathbb{E}_{q(\epsilon)}$ is the expectation over random Gaussian noise ϵ , we sample ϵ , and then compute just the inside of the expectation to update the parameters. This is a special case of Monte Carlo sampling where

the sample size is 1 for each update. This approach works, as it acts as stochastic gradient descent. It is important to note that though \mathbf{w} is a vector of weights, each weight is independent and thus only pointwise operations take place.

The update happens in batches, where the batch size can be the entire training dataset. For each batch, we draw a single sample \mathbf{w} from the posterior distribution that is then used to update the same posterior. We start by sampling from a unit Gaussian, and then shifting by parameters θ to obtain a sample of weights:

$$\epsilon \sim \mathcal{N}(0, I) \quad (3.14)$$

$$\mathbf{w} = \mu + \log(1 + \exp(\rho)) \cdot \epsilon \quad (3.15)$$

With $f(\mathbf{w}, \theta)$ defined above, and, since $\frac{\partial \mathbf{w}}{\partial \mu} = 1$ and $\frac{\partial \mathbf{w}}{\partial \rho} = \frac{\epsilon}{1 + \exp(-\rho)}$, we calculate the following gradients:

$$\Delta_\mu = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \mu} \quad (3.16)$$

$$\Delta_\sigma = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} \frac{\epsilon}{1 + \exp(-\rho)} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \rho} \quad (3.17)$$

Finally, we update our parameters:

$$\mu \leftarrow \mu - \alpha \Delta_\mu \quad (3.18)$$

$$\rho \leftarrow \rho - \alpha \Delta_\sigma \quad (3.19)$$

Let us look at how we compute the gradients Δ_μ and Δ_σ . ϵ is a known sample, and μ and ρ are fixed from the previous state of the variational posterior. We need only find $\frac{\partial f(\mathbf{w}, \theta)}{\partial \mu}$, $\frac{\partial f(\mathbf{w}, \theta)}{\partial \rho}$, and $\frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}}$.

3.4 The Loss Function

We interpret $f(\mathbf{w}, \theta) = \log q(\mathbf{w}|\theta) - \log P(\mathbf{w}) - \log P(\mathcal{D}|\mathbf{w})$ as follows. Since q is the approximating variational posterior, $\log q(\mathbf{w}|\theta)$ penalizes samples that are far from the posterior. $P(\mathbf{w})$ is the

prior, and thus acts as a regularizing term on the weights. Finally, $\log P(\mathcal{D}|\mathbf{w})$ is a data dependent term that corresponds to the prediction error. With a Gaussian prior and Gaussian posterior, we know that:

$$q(\mathbf{w}|\theta) \sim \mathcal{N}(\mu_{\text{posterior}}, \sigma_{\text{posterior}}) \quad (3.20)$$

$$P(\mathbf{w}) \sim \mathcal{N}(\mu_{\text{prior}}, \sigma_{\text{prior}}) \quad (3.21)$$

To find their partials, we need only differentiate the normal PDF.

The data dependent term requires more consideration. $P(\mathcal{D}|\mathbf{w}) = \prod_{i=1}^n P(y_i|x_i, \mathbf{w})$ is the probability of obtaining the data with given weights \mathbf{w} and n samples in the batch. Then,

$$\log P(\mathcal{D}|\mathbf{w}) = \sum_{i=1}^n \log P(y_i|x_i, \mathbf{w}) \quad (3.22)$$

We fix $P(y_i|x_i, \mathbf{w}) = P(y_i)$ to be a Gaussian, and consider its PDF:

$$p(y_i) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2} (y_i - \hat{y}_i)^2\right) \quad (3.23)$$

$$\log p(y_i) = \log \frac{1}{\sigma\sqrt{2\pi}} - \frac{1}{2\sigma^2} (y_i - \hat{y}_i)^2 \quad (3.24)$$

$$= C - \gamma (y_i - \hat{y}_i)^2 \quad (3.25)$$

The σ term for the probability of the data, as we will see later, corresponds to the aleatoric noise on the data. For now, we keep it fixed at an arbitrary value with no loss to the model. Then, for gradient descent, taking the partial derivative w.r.t. the output \hat{y}_i gives:

$$-\frac{\partial \log p(y_i)}{\partial \hat{y}_i} = -\frac{1}{\sigma^2} (y_i - \hat{y}_i) \quad (3.26)$$

Thus,

$$\frac{\partial \log p(y_i)}{\partial \mathbf{w}} = \frac{\partial \log p(y_i)}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial \mathbf{w}} \quad (3.27)$$

$$= \frac{1}{\sigma^2} (y_i - \hat{y}_i) \cdot \frac{\partial \hat{y}_i}{\partial \mathbf{w}} \quad (3.28)$$

and $\frac{\partial \hat{y}_i}{\partial \mathbf{w}}$ is found by differentiating through the network. Since \mathbf{w} is a fixed sample, the $P(\mathcal{D}|\mathbf{w})$ term has no dependence on μ or ρ .

Now, to calculate $\frac{\partial f(\mathbf{w}, \theta)}{\partial \mu}$ and $\frac{\partial f(\mathbf{w}, \theta)}{\partial \rho}$ we only need to differentiate $\log q(\mathbf{w}|\theta)$. This is because both $P(\mathcal{D}|\mathbf{w})$ and $P(\mathbf{w})$ have no dependence on μ or ρ since \mathbf{w} now represents a fixed sample.

3.5 Connection to VAEs

A variational autoencoder (VAE) is similar to the BNN in that its latent space is represented as a Gaussian with a prior. Through training, we seek to minimize the KL divergence between the latent Gaussian and a true, unknown distribution using Bayes theorem.

The training process of a VAE is equivalently the variational method outlined by Kendall and Gal for BNNs. [26] That is, the weights in the latent space are set to be Gaussians whose mean and standard deviation are trained by backpropagation. If we look at the training diagram produced by Rocca [26], one might think that the Bayesian layer is different from the BNN in this way: instead of the parameters of the Bayesian layer being set in training, they are decided by a split head from the previous layer. Both of these configurations are in fact congruent. In the method of Kendall and Gal, a weight w is sampled from a Gaussian with fixed parameters μ and σ and multiplied by an input x to get output y ; where ϵ is Gaussian noise.

$$y = xw = x(\mu + \sigma \cdot \epsilon) \tag{3.29}$$

$$= x\mu + (x\sigma) \cdot \epsilon \tag{3.30}$$

In the split head approach (described later in detail, in Section 5.3), input x is first split into two by trained weights w_1 and w_2 , producing the parameters of the Gaussian.

$$\mu_s = w_1x \tag{3.31}$$

$$\sigma_s = w_2x \tag{3.32}$$

Then, the Gaussian is sampled.

$$y = \mu_s + \sigma_s \cdot \epsilon \tag{3.33}$$

$$= w_1 x + (w_2 x) \cdot \epsilon \tag{3.34}$$

Now it is easy to see that the split head weights are exactly the trained parameters in the first approach. That is,

$$\mu = w_1 \tag{3.35}$$

$$\sigma = w_2 \tag{3.36}$$

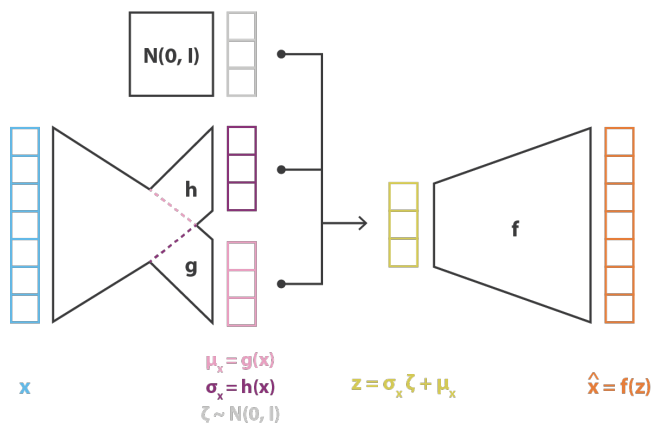


Figure 3.2: The latest space in the VAE, as shown in this figure reproduced from [26], is trained in the same way as in the work of Blundell et al. [4].

To make a traditional autoencoder a VAE, you only need to add a linear BNN layer right after the encoder portion (and thus representing the latent space). The parameters of this BNN layer are adjusted during training and define the distribution to be sampled from in the generative mode.

As we see in figure 3.2, Rocca [26] implements the same reparameterization technique described in the work of Blundell et al. [4], but only for the latent space. They suggest doing this for each weight in the model.

An and Cho propose the use of VAEs for anomaly detection (which is a form of epistemic uncertainty), and apply this to the MNIST dataset. [1] Instead of using the reconstruction error directly as the uncertainty metric, they propose using the reconstruction probability. They calculate the reconstruction error in much the same way we predict with a BNN; that is, they take the Monte Carlo estimate of the log probabilities. In practice, this means taking the mean of a set a distances that are computed for independently sampled outputs of the VAE.

Chapter 4

Implementations

4.1 Analytical Approach

The central question regarding overall aleatoric uncertainty in the model is the following: How can one create a metric to measure the overall variation of the model, independent of a specific input. Just as a traditional neural network is a function with a set of inputs and outputs, a BNN is simply a function of random variables—specifically Gaussians. The trained model as a whole is itself a random variable, and its overall variance can be calculated. This is made easier due to the fact that each weight produces a value that is uncorrelated with other weights, when it is sampled.

To make this clearer, consider just a single node of a single, fully-connected layer of a BNN defined by:

$$Y_i = f(y_i) = f(x_1w_1 + x_2w_2 + \cdots + x_nw_n) \tag{4.1}$$

Where f is the activation function, x_i is the layer input, and w_i is the layer weights. Further,

$w_i \sim N(\mu_i, \sigma_i^2)$. If we let $f(y_i) = cy_i$, then

$$\text{Var}[f(y_i)] = \text{Var}[cy_i] \tag{4.2}$$

$$= c^2 \text{Var}[y_i] \tag{4.3}$$

$$= c^2 \text{Var}[x_1 w_1 + x_2 w_2 + \dots + x_n w_n] \tag{4.4}$$

If we consider x_i a fixed value, then

$$\text{Var}[f(y_i)] = c^2(x_1^2 \text{Var}[w_1] + x_2^2 \text{Var}[w_2] + \dots + x_n^2 \text{Var}[w_n]) \tag{4.5}$$

$$\text{Var}[f(y_i)] = c^2(x_1^2 \sigma_1^2 + x_2^2 \sigma_2^2 + \dots + x_n^2 \sigma_n^2) = c^2 \sum_{i=1}^n x_i^2 \sigma_i^2 \tag{4.6}$$

We see that the variance has a dependency on the layer inputs as well as the weight variances, and the variance of the BNN will ultimately have a dependency on the the model inputs. Thus, this measure of variance will ultimately be data dependent and correspond to heteroscedastic aleatoric uncertainty.

4.2 Regression Implementation

For our first implementation, we build a single layer, sixth-order polynomial regression model.

$$y_i = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + w_5 x^5 + w_6 x^6 \tag{4.7}$$

Data has been generated by a fourth-order polynomial in the interval [0,1] with added Gaussian noise that is non-uniform. We show this in figure 4.1.

We aim to train our variational posterior weight distributions $\mathbf{w} = [w_0, w_1 + \dots + w_6]$ with Bayes by Backprop. For starters, we let our prior on our weights be an i.i.d. unit Gaussian, and we initialize the posterior as follows.

$$P(\mathbf{w}) \sim \mathcal{N}(\mu_{\text{prior}} = 0, \sigma_{\text{prior}} = I) \tag{4.8}$$

$$q(\mathbf{w}|\theta) \sim \mathcal{N}(\mu_{\text{posterior}} = 0.1 \cdot I, \sigma_{\text{posterior}} = 0.1 \cdot I) \tag{4.9}$$

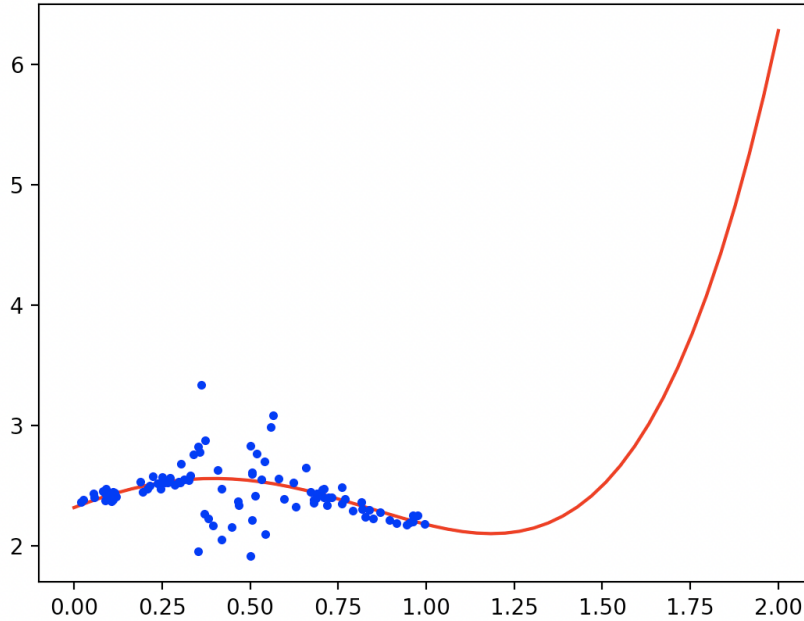


Figure 4.1: Data (blue) generated from a fourth-order polynomial (red) with non-uniform Gaussian noise.

We train as above, with a batch size of the whole data set, for 10,000 iterations. Take special note of $\frac{\partial \hat{y}_i}{\partial \mathbf{w}}$. This term is specific to the model definition and is at the core of backpropogation. For this case,

$$\frac{\partial \hat{y}_i}{\partial w_i} = x^i \tag{4.10}$$

$$\frac{\partial \hat{y}_i}{\partial \mathbf{w}} = [1, x, x^2 + \dots + x^6] \tag{4.11}$$

Once we have trained our model, we can predict on it either by taking the expectation on the weights, or by taking multiple random samples from the model for a given input, and constructing a confidence interval. Here, we take 100 samples for each input, and construct an 80% confidence interval as seen in figure 4.2.

Later, we will look at data-dependent uncertainty metrics on our model.

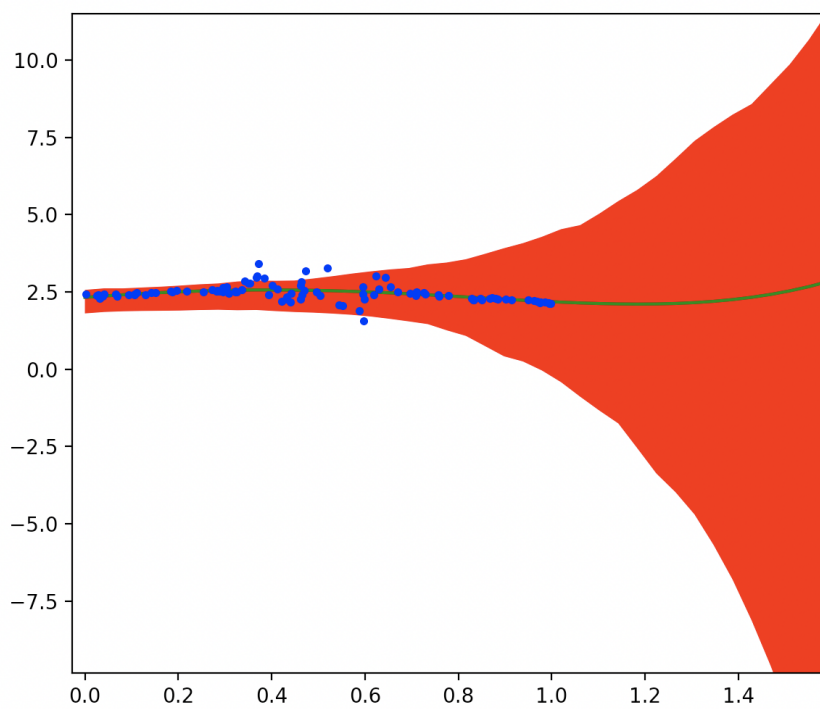


Figure 4.2: 80% confidence interval from our model (red) to the data (blue) generated from a fourth-order polynomial (green) with non-uniform Gaussian noise.

4.3 MNIST implementation

A Bayesian neural network (BNN) replaces each weight of the model with a random variable. We can approximate the weight distributions with a two parameter normal distribution through variational learning. Training on the model thus adjusts the mean and variance of each weight by minimizing the Kullback–Leibler divergence between the approximating normal distribution and the true, unknown distribution.

The Kullback-Leibler divergence is a measure between two probability distributions P and q on the same probability space X and is defined in equation 3.5.

Predicting with a BNN can be deterministic (by taking the expected value of the weights to serve as fixed weights), or stochastic (by randomly sampling from the weights). Blundell et al. give a more in-depth overview of BNNs [4].

A BNN can produce both a prediction and an uncertainty metric; the later is found by looking at the variance of prediction from a set of samples from the stochastic model. Kwon et al. propose a method to distinguish between epistemic and aleatoric uncertainty in a BNN [20]. Experimental results show that a BNN does very well at giving an error metric that encompasses both types of uncertainty. But, the division between the two types can seem unclear at first inspection.

Kwon et al. give a formula that, applied to a BNN, will give error metrics for both epistemic and aleatoric uncertainty metrics separately. The aleatoric uncertainty metric seeks to give a measure of the inherent noisiness in the dataset, while the epistemic uncertainty metric seeks to quantify the level of confidence on a particular domain of data.

Shridhar et al. provide PyTorch code for implementing a BNN and predicting both kinds of uncertainty as detailed by Kwon et al. [31]. For this experiment, we train a Bayesian-CNN (BCNN) on MNIST data and predict on both MNIST and non-MNIST data. We ran the model on 50,000 MNIST and 50,000 non-MNIST samples, and recorded the uncertainty metric of each. MNIST samples are split into correct and incorrect groupings. As the model has a high level of accuracy, the account of incorrect predictions is relatively small. In figure 4.3, we see a good division between the “Correct MNIST” distribution, and the other distributions.

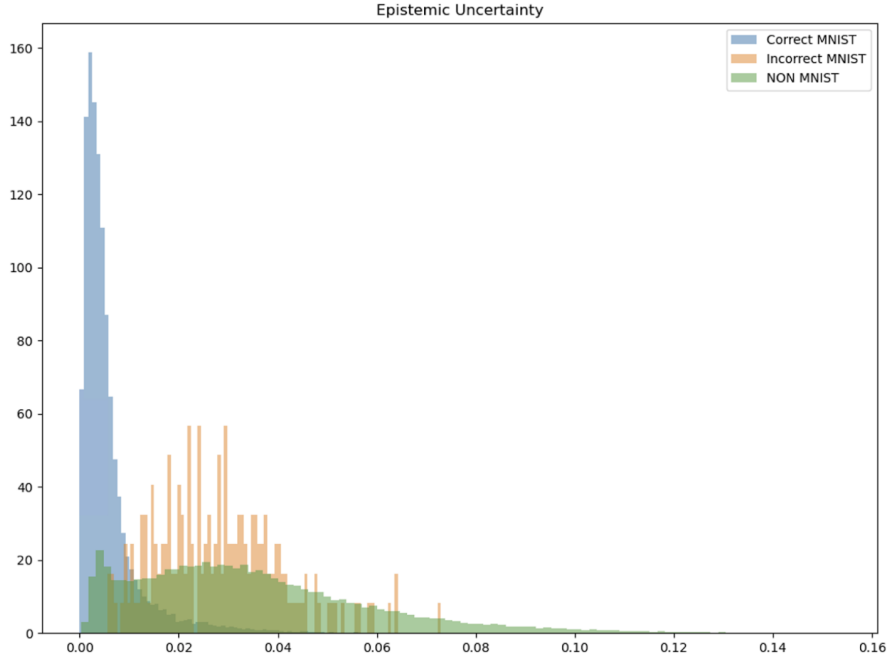


Figure 4.3: Histogram of epistemic uncertainty metrics on 50,000 samples from MNIST and non-MNIST as defined in the second part of equation 4.12 below.

The MNIST dataset is a large set of 28 by 28 pixel, greyscale, hand drawn, labeled characters (zero to nine[0-9]) that is a common starting point for ML based tasks. The set includes 60,000 training and 10,000 test images. [21] Here, we use the MNIST dataset to explore BNNs.

The formulation for both uncertainties by Kwon et al. is completely dependent on the sampled outputs of the BNN, and is given by the diagonal matrix element:

$$U_k(x_i) = \left[\underbrace{\frac{1}{T} \sum_{t=1}^T \text{diag}(\hat{p}_t) - \hat{p}_t^{\otimes 2}}_{\text{aleatoric}} + \underbrace{\frac{1}{T} \sum_{t=1}^T (\hat{p}_t - \bar{p})^{\otimes 2}}_{\text{epistemic}} \right]_{kk} \quad (4.12)$$

for class k and data input x_i , where $\bar{p} = \frac{1}{T} \sum_{t=1}^T \hat{p}_t$. \hat{p}_t is the vector of softmax outputs of the BNN that have been passed through the final Softmax function, T is the number of predictions made on the same input data, and $v^{\otimes 2} = vv^T$. The weights of the BNN are sampled each time randomly from their respective distribution.

We can see in this formulation that the aleatoric uncertainty is an inverse measure of how close the

prediction is to zero or one. When closer to 0.5, this metric is high. The epistemic uncertainty, on the other hand, is a measure of the variance of the predictions irrespective of their mean.

To understand the above formulation, consider a class that is undersampled. One would expect the variance of that class to be high and thus high epistemic uncertainty. Now consider a class that is well represented, but is very noisy. One would expect the model weights to converge to a value, but the prediction to be closer to 0.5. These respectively correspond to a high epistemic and aleatoric uncertainty above.

To apply this definition of uncertainty, we consider the metric on only the final prediction class. The epistemic portion then becomes the mean squared difference between the predictions in the prediction-mean—that is the variance of prediction. The aleatoric portion then becomes the average value of $\hat{p}_t - \hat{p}_t^2$. Considering that \hat{p}_t is a prediction from $[0,1]$, this value approaches 0 as the prediction approaches the endpoints, and is highest when the prediction is closest to 0.5. In this way, the aleatoric uncertainty is just looking at how “confident” the softmax output is. This can be a good classifier for incorrectly predicted, in-class data as shown by Ståhl et al. [32].

These uncertainty metrics might be improved by considering not only the variation of the model output (the prediction uncertainty), but also the variation of the internal weights (the model variability). Since the model weight parameters are fixed after training, such an analysis of its weight variability should be done without respect to any specific input. This can give a better aleatoric uncertainty metric, and thus tell the model engineer when the model has trained on sufficient data. And in some cases, it might tell the data engineer when the aleatoric uncertainty is no longer improving with the addition of more data (such might be the case with noisy, or inconsistent data).

4.4 Characterizing Datasets

As we have seen in section 4.1, the analytical variance of the Bayesian model presented is wholly dependent on the mean vector and covariance matrix of the model inputs. Thus, it would be reasonable to think that one might characterize a particular dataset by only its mean and covariance, and that would be enough to assess model certainty on the whole data set. This method might work particularly well if you have access to a large dataset that is unseen by the model, but where running each sample through the model and measuring certainty would be too computationally expensive. The idea is as follow:

1. For each dataset (for example, training subset and unseen dataset) perform any preprocessing steps and compute the mean and covariance of the data (that is, the mean and covariance on what would be the inputs to the model). Thus you have a multivariate random normal distribution that is representative of the dataset.

$$X \sim \mathcal{N}(\mu, \Sigma) \tag{4.13}$$

2. For each multivariate random normal distribution, sample some N times and pass each sample through the BNN, computing the uncertainty metric as described above. Let $U(x_i)$ be the uncertainty metric for data x_i . Take an average of the uncertainties across the samples. This serves as the uncertainty metric for the model on the dataset x_i .

$$\frac{1}{N} \sum_{i=1}^N U(x_i) \tag{4.14}$$

MNIST Data

We can classify the MNIST data of size $n \times n$ with the following mean and covariance matrix. The $n \times n$ mean matrix is shown in figure 4.4, and the variance matrix in figure 4.5. The full covariance matrix is $n \times n \times n \times n$.

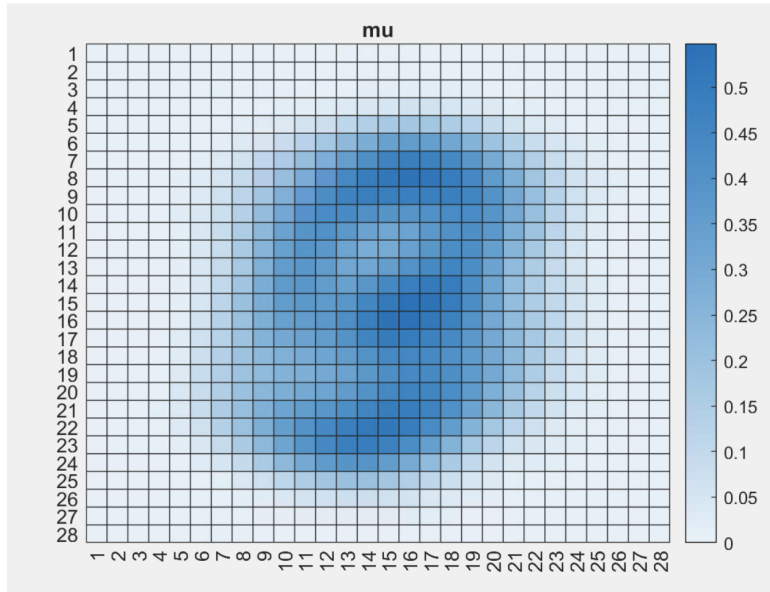


Figure 4.4: MNIST mean values on a per-pixel basis, normalized from [0,1].

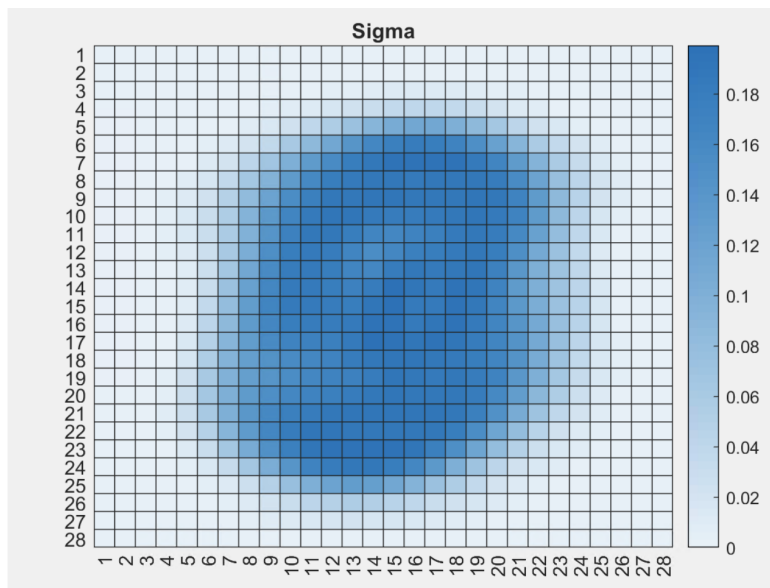


Figure 4.5: MNIST Variance Matrix, normalized from [0,1].

Having calculated μ and σ for our dataset (MNIST digits), we can model with a multivariate normal distribution, and sample from this distribution. Calculating the certainty metric on these samples should give us a good idea of the overall uncertainty on the dataset as a whole.

In figure 4.6, we see 100 random samples from our multivariate random normal distribution that represents the MNIST dataset. Many of these samples have characteristics of numerals, and some could even be interpreted as a specific numeral. Overall, it is not a bad characterization of the MNIST dataset in a compact form.

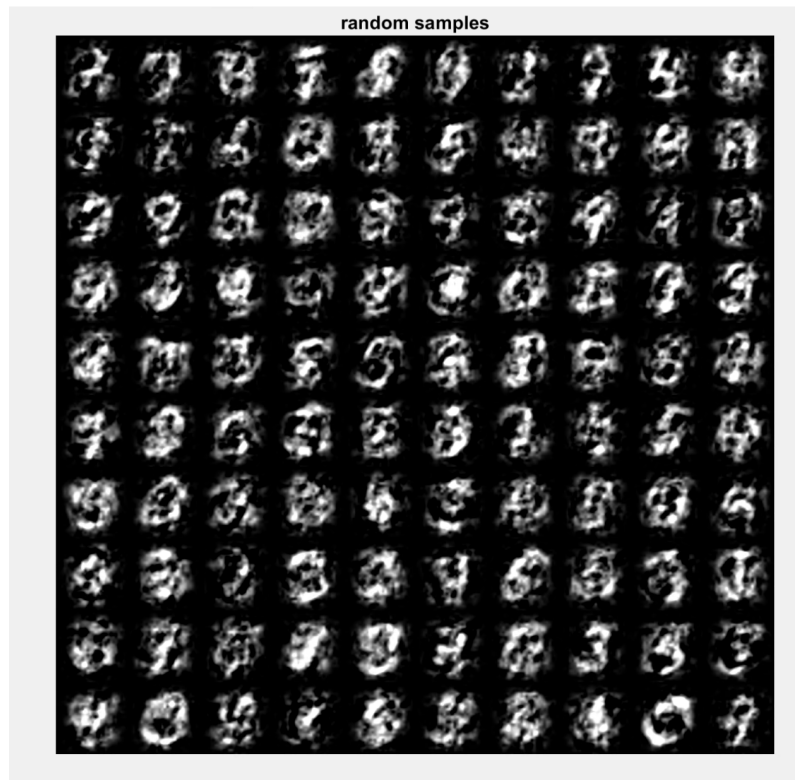


Figure 4.6: 100 random samples from the multivariate normal distribution that characterizes the MNIST dataset.

Flower images

Using a small dataset of 633 images of daisies cropped to 128x128 pixels, we can apply the same technique.



Figure 4.7: Flower data set.

Trying the same approach on an image set of cropped flowers, shown in figure 4.7, does not give as good results. As the flowers are not centered, and the cropping is not consistent, the mean (figure 4.8) and variance matrix (figure 4.9) appear pretty flat, and the random samples from the characterization are very noisy. We see examples of these random samples in figure 4.10. The samples have no resemblance to flowers.

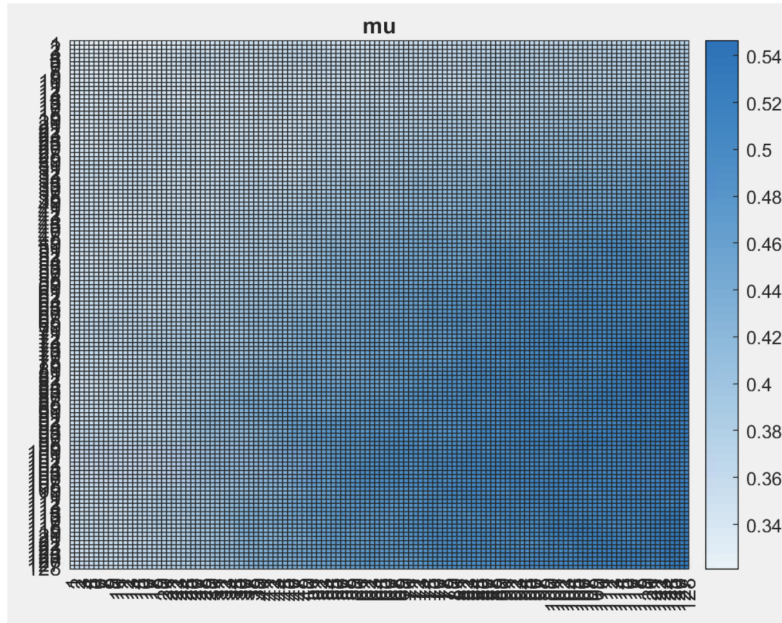


Figure 4.8: Mean values for the flower dataset.

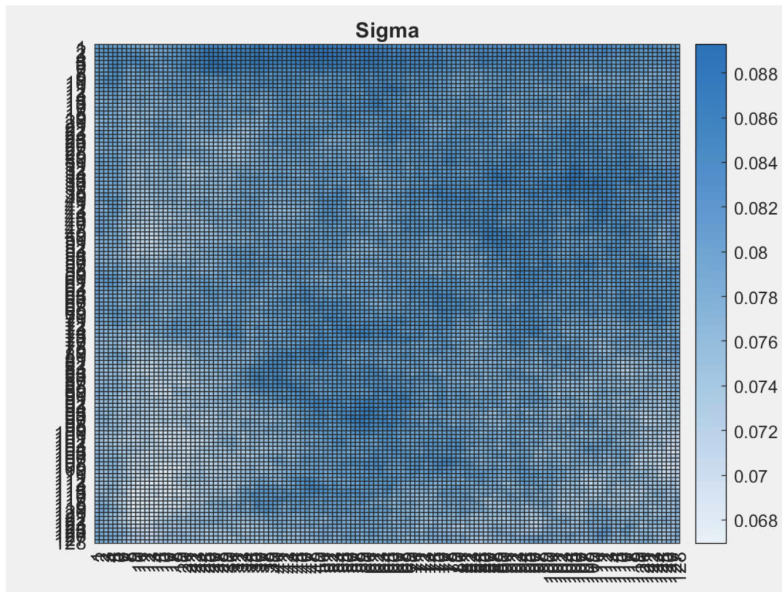


Figure 4.9: Variance matrix for flower dataset.

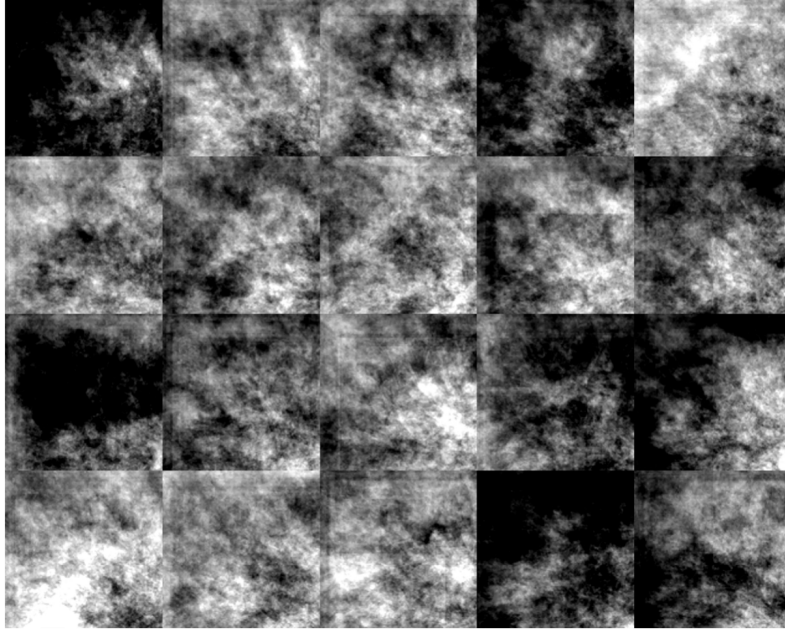


Figure 4.10: Random samples from the flower multivariate random normal distribution.

From these two experiments, it is evident that this method is more effective on clean datasets such as MNIST than on other datasets that bear more complexity.

Chapter 5

Measuring Uncertainty

5.1 Epistemic Uncertainty

Kwon et al. [20] propose a data dependent metric for epistemic uncertainty that is exactly the variance of a set of predictions from a single input to the model. Recall that a BNN can be used for inference in a deterministic mode, by using the weight means, or in a random mode, by sampling from the weights. By taking a set of samples, and thus a set of predictions from an input, we can take the variance and get a metric for epistemic uncertainty.

$$U_{\text{epistemic}} = \frac{1}{T} \sum_{t=1}^T (\hat{p}_t - \bar{p})^2 \quad (5.1)$$

Here, T predictions \hat{p}_t are sampled from the model, and \bar{p} is the mean of these predictions. For classification, \hat{p}_t is the output of the final softmax layer; but for our regression case, it is the raw prediction.

In Figure 5.1, we implement this metric on our polynomial model from Section 4.2. The data-dependent epistemic uncertainty shown as a blue line is low when close to the data, and increases rapidly past $x = 1.0$, where the data points end. Note, furthermore, that although there is more noise on the interval $[1/3, 2/3]$, we do not see a higher epistemic uncertainty. This is expected, as noise in data is an example of aleatoric uncertainty.

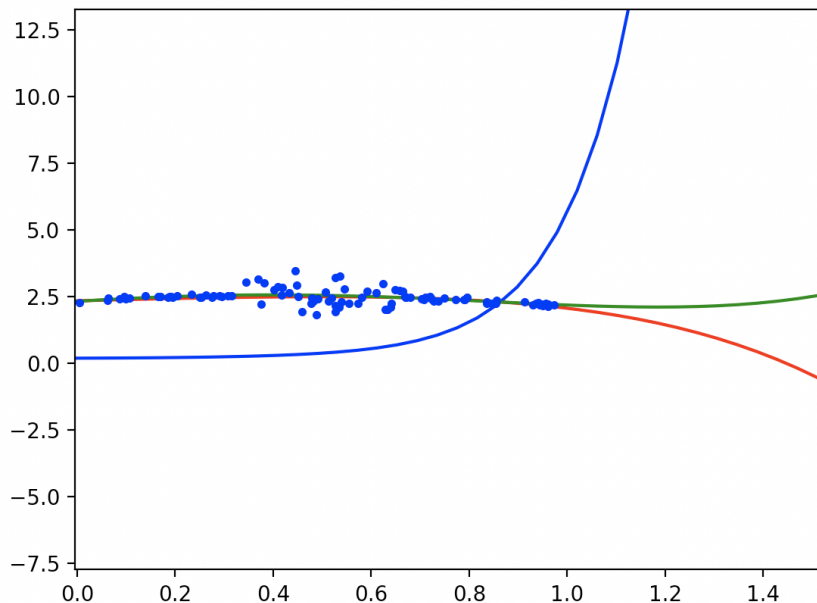


Figure 5.1: From data (blue dots) generated from the green curve, we plot the prediction in red and the epistemic uncertainty in blue.

5.2 Homoscedastic Aleatoric Uncertainty

Homoscedastic uncertainty assumes constant aleatoric noise across the data, thus we might only obtain a scalar estimate of such uncertainty. Kendall and Gal present the distinction between this kind and heteroscedastic uncertainty. [18] On our single layer model, we can measure the homoscedastic uncertainty by taking the mean of the variances of each weight.

$$U_{\text{aleatoric}} = \frac{1}{T} \sum_{t=1}^T \hat{\sigma}_t^2 \quad (5.2)$$

For our model, we get a measure of 1.06302 for homoscedastic uncertainty.

5.3 Heteroscedastic Aleatoric Uncertainty

Kendall and Gal propose a method to build a BNN that predicts not only epistemic uncertainty, but also heteroscedastic (data-dependent) aleatoric uncertainty. [18] They build a BNN with a split head, where one head gives the prediction and the other gives the heteroscedastic model uncertainty

for the given input. That is, for sampled weights $\hat{\mathbf{w}} \sim q(\mathbf{w})$, the model gives:

$$[\hat{\mathbf{y}}, \hat{\sigma}^2] = f^{\hat{\mathbf{w}}}(\mathbf{x}) \quad (5.3)$$

With this split head, Kendall and Gal then propose the following loss function, while omitting the weight normalization term for brevity (and to prevent ambiguity):

$$\mathcal{L}_{BNN}(\theta) = \frac{1}{N} \sum_i \frac{1}{2\hat{\sigma}_i^2} \|\hat{\mathbf{y}}_i - \mathbf{y}_i\|^2 + \frac{1}{2} \log \hat{\sigma}_i^2 \quad (5.4)$$

Here, $\hat{\sigma}_i$ is the data dependent output of the split head, not a hyperparameter as we saw above.

We can easily update our loss with our above construction. Recall our loss function: $f(\mathbf{w}, \theta) = \log q(\mathbf{w}|\theta) - \log P(\mathbf{w}) - \log P(\mathcal{D}|\mathbf{w})$. We pick up from our exploration of the last data term in section 3.4, $P(\mathcal{D}|\mathbf{w}) = \prod_{i=1}^n P(y_i|x_i, \mathbf{w})$. Also recall: $P(y_i|x_i, \mathbf{w}) = P(y_i)$ is fixed to be a Gaussian, and consider its PDF keeping in mind that both $\hat{\sigma}$ and \hat{y} are now data dependent:

$$p(y_i) = \frac{1}{\hat{\sigma}_i \sqrt{2\pi}} \exp\left(-\frac{1}{2\hat{\sigma}_i^2} (y_i - \hat{y}_i)^2\right) \quad (5.5)$$

$$-\log p(y_i) = \frac{1}{2\hat{\sigma}_i^2} (y_i - \hat{y}_i)^2 + \log \hat{\sigma}_i \sqrt{2\pi} \quad (5.6)$$

and since

$$\log P(\mathcal{D}|\mathbf{w}) = \log \prod_{i=1}^n P(y_i|x_i, \mathbf{w}) = \sum_{i=1}^n \log p(y_i) \quad (5.7)$$

Our final loss term becomes:

$$-\log P(\mathcal{D}|\mathbf{w}) = \sum_{i=1}^n \log \hat{\sigma}_i \sqrt{2\pi} + \frac{1}{2\hat{\sigma}_i^2} (y_i - \hat{y}_i)^2 \quad (5.8)$$

We have found above, and still holds true that:

$$\frac{\partial \log p(y_i)}{\partial \hat{y}_i} = \frac{1}{\hat{\sigma}_i^2} (y_i - \hat{y}_i) \quad (5.9)$$

We then just need to find the partial derivative w.r.t. $\hat{\sigma}$:

$$-\frac{\partial \log p(y_i)}{\partial \hat{\sigma}} = \frac{1}{\sigma} - \frac{1}{\sigma^3} (y_i - \hat{y}_i)^2 \quad (5.10)$$

We will follow the lead of Kendall and Gal; and, instead of predicting the standard deviation, we will predict the log variance $s_i := \log \hat{\sigma}_i^2$. Recalling our data distribution:

$$p(y_i) = \frac{1}{\hat{\sigma}_i \sqrt{2\pi}} \exp\left(-\frac{1}{2\hat{\sigma}_i^2} (y_i - \hat{y}_i)^2\right) \quad (5.11)$$

$$-\log p(y_i) = \frac{1}{2} \exp(-s_i) (y_i - \hat{y}_i)^2 + \frac{1}{2} s_i + \log \sqrt{2\pi} \quad (5.12)$$

Differentiating w.r.t. \hat{y}_i and s yields:

$$\frac{\partial \log p(y_i)}{\partial s} = \frac{1}{2} \exp(-s_i) (y_i - \hat{y}_i)^2 - \frac{1}{2} \quad (5.13)$$

$$\frac{\partial \log p(y_i)}{\partial \hat{y}_i} = \exp(-s_i) (y_i - \hat{y}_i) \quad (5.14)$$

Chapter 6

Applications to Aerial Segmentation

In their paper titled Aerial Imagery Pixel-level Segmentation, Heffels and Vanschoren look at applying current state of the art segmentation models to aerial segmentation problems. [14] Specifically, they use the DeepLabv3+ Xception65 architecture on the DroneDeploy dataset [9] and achieves an mIoU of 70%. They also benchmark the use of the more fundamental U-Net architecture on the same data.

Aerial Segmentation is a pertinent topic that has broad reaching applications, ranging from understanding deforestation, biological diversity, and poverty [14] to developing high definition lane-level maps for autonomous driving. The latter involves processing segmentation results over road classes in order to generate georeferenced map vector data.

Annotation is done at large scale, and suffers from both aleatoric and epistemic uncertainty. Aleatoric uncertainty often arises when there is a discrepancy between annotations which introduces noise. For example, some annotators might label an airport runway as road, while others will annotate as background. Epistemic uncertainty arises whenever we take a trained model and try to predict on new, unseen data. Different geographic regions offer different terrain, and must be annotated.

For our experiments here, we will stick with the U-Net architecture (detailed in section 6.2) on the DroneDeploy dataset (detailed in section 6.1), with the simple goal of modifying this architecture to use BNN layers and thus produce aleatoric and epistemic uncertainty metrics with each prediction. With this goal, we aim to show the feasibility of modifying almost any deep learning architecture to become a BNN by replacing traditional layers with Bayesian layers. We use a PyTorch implementation of the U-Net architecture found here [25] on the DroneDeploy dataset outlined here [9].

We have chosen the scalar metric **mIoU**, also known as the Jaccard distance, to benchmark our results, as it is well suited for segmentation tasks. [33] For a given semantic class where A is the ground truth and B is the prediction, the intersection over union (IoU) is defined as:

$$\text{IoU} = \frac{|A \cap B|}{|A \cup B|} \quad (6.1)$$

Where $|\cdot|$ is the cardinality. Now, **mIoU** is simply the mean IoU across all n classes:

$$\text{mIoU} = \frac{1}{n} \sum_{i=1}^n \text{IoU}_i \quad (6.2)$$

6.1 The DroneDeploy Dataset

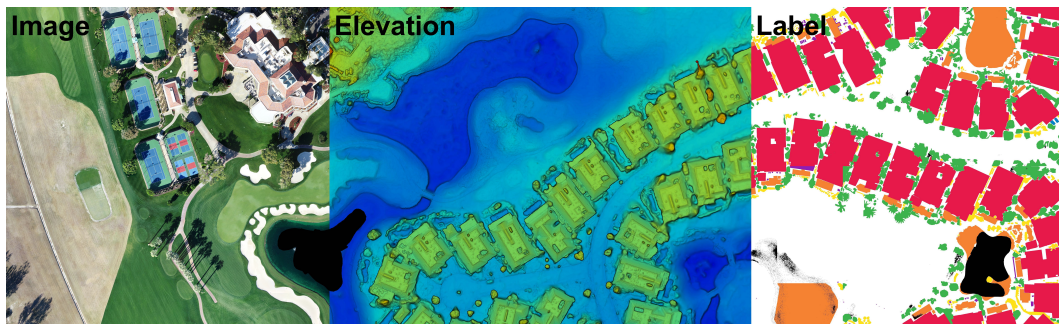


Figure 6.1: DroneDeploy dataset example. [9]

The DroneDeploy dataset [9] is a collection of aerial images, DEMs (Digital Elevation Models), and associated semantic labels. DroneDeploy offers a web-based photogrammetric processing toolbox that is marketed for drone imagery. Using geo-referenced aerial imagery, an orthoimage and DTM

is produced. An orthoimage is a geometrically corrected (orthorectified) photo such that scale is uniform across the image. In addition, for most applications, precise geo-locations are given for each pixel. More specifically, a transformation matrix is provided to convert from pixel-coordinates to geo-coordinates or vice-versa, where the geo-coordinates are in a spatial reference system of your choosing—the most common of these is WGS84, or the “standard” latitude/longitude.

The DroneDeploy dataset offers a “small” and “medium” size dataset. We use the “medium” dataset with 55 geoTiffs and associated, pixel-level semantic labels. The codebase found on GitHub as provided by DroneDeploy [9] will download the data and create a tiled dataset with image and label chips of size 300x300 pixels. The elevation data is left to future implementation for both their example and our work here.

In continued research there is potential to leverage the elevation data (in either a raw or normalized form) as a separate image channel to improve semantic predictions. This is evident by the fact that buildings are often elevated above their surrounding, water and roads are fairly flat, and vehicles have some curvature.

The semantic labels and associated (Blue, Green, Red) colors included in the dataset are:

(075, 025, 230) : *BUILDING*

(180, 030, 145) : *CLUTTER*

(075, 180, 060) : *VEGETATION*

(048, 130, 245) : *WATER*

(255, 255, 255) : *GROUND*

(200, 130, 000) : *CAR*

(255, 000, 255) : *IGNORE*

For our experiments, we have changed ground class to (0, 75, 150) to better differentiate it.

The dataset offers a variety of terrains that include residential, agricultural, and commercial use areas. Ground and Vegetation classes are the most represented classes at 37.7% and 10.43% respec-

tively. The full class distribution can be seen in table 6.1. In addition to increasing the amount of data, prediction quality might be improved by better class differentiation. For example, the ground class can be divided into road and general terrain.

Using calculations from Heffels [14], we have computed the class distribution of just the classes that are in the sampled data. There is a “0:Ignore” class that makes up 42.7% of the the raw images, but this area is ignored by the sampler.






Class	Color code	Proportion of sampled pixels
1: Building	Red 	9.8%
2: Clutter	Purple 	3.5%
3: Vegetation	Green 	18.2%
4: Water	Orange 	2.1%
5: Ground	White	65.8%
6: Car	Blue 	0.7%

Table 6.1: Class distribution of DroneDeploy dataset.

To address the problem of class imbalance, Heffels utilized the focal loss introduced by Linn et al. [22], but has no measurable performance boost. This is worth further exploration in the future; since, as we will see, the model performance on the lowest represented classes can be quite poor.

6.2 U-Net Architecture

The U-Net architecture introduced by Ronneberger et al., shown in figure 6.2, is a landmark, fully convolutional architecture that won the 2015 ISBI cell tracking challenge with a large margin. [27] It is able to perform well with a relatively small amount of training images. This network laid the foundation for today’s state of the art DeepLabv3 architecture.

The U-Net architecture has similarities to a fully convolutional autoencoder in that it has a down-sampling and an upsampling component; but, with the addition of copy and crop skip connections. These skip connections bring relevant spatial information from preceding layers to their upsampled counterparts.

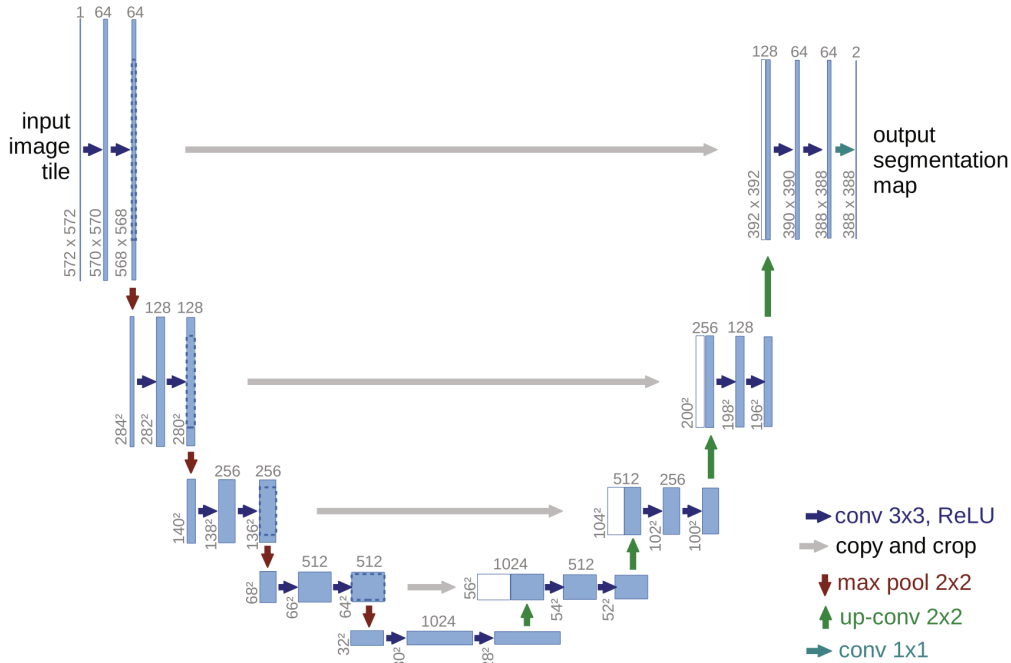


Figure 6.2: The landmark U-Net architecture by Ronneberger et al. Figure reproduced from [27].

There are a variety of ways to upsample an image layer. In the U-Net architecture, this is done with linear or bilinear interpolation. DeepLabv3 uses dilated convolutional filters that gets spatial information from pixels 1,2, or 3 neighbors away. One can imagine a 3x3 grid that is dilated such that there are $n - 1$ pixel gaps between each neighboring pixel for dilation rate D . Finally, Chen et al. proposed an upsampling method called Atrous Spatial Pyramidal Pooling (ASPP). [5] This method concatenates a collection of upsampled images with different dilation rates to give multiple fields-of-view. Hefels explores these different upsampling methods in more detail. [14]

Our U-Net architecture [25] has a straightforward PyTorch implementation, excerpts of which are given in Appendix A.1.

We have run the U-Net architecture on the DroneDeploy dataset and achieved a validation mIoU of 0.70, which is comparable to results of Hefels et al. [14] with performance that matches that of DeepLabv3. Our validation mIoU, train loss, and Dice loss are plotted in figure 6.3. This shows that for a smaller dataset such as this one, there may not be much added benefit to DeepLabv3 when compared to the simpler U-Net. U-Net has been shown to work well on smaller datasets; this is partly explained due to its smaller number of trained parameters.

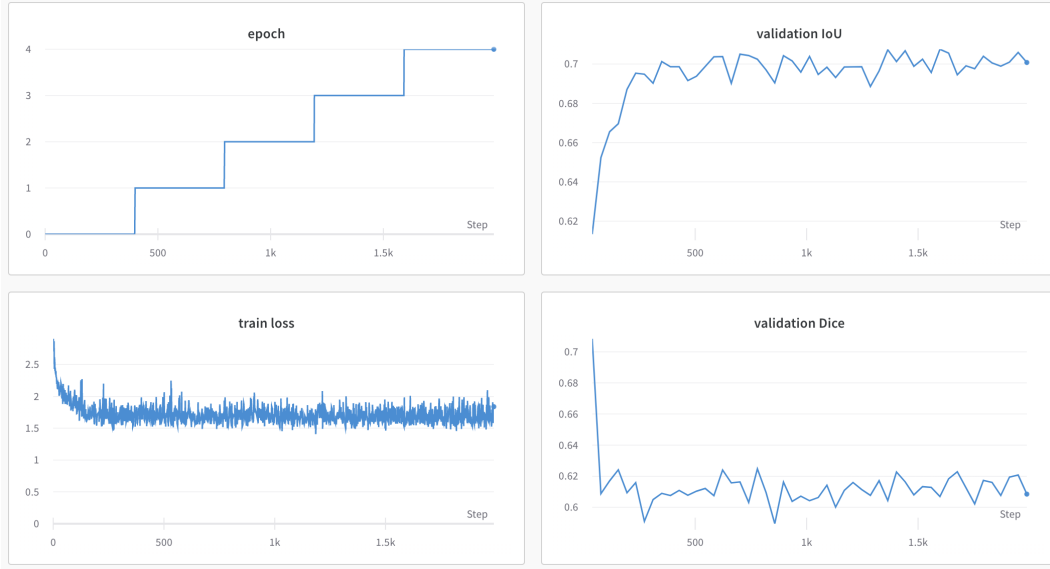


Figure 6.3: Training results of the U-Net architecture on the DroneDeploy dataset. We see fairly quick convergence to the final mIoU.

6.3 Bayesian Aerial Segmentation: A Bayesian U-Net

In order to build a Bayesian U-Net, we have replaced the traditional convolutional layers with Bayesian layers as proposed in the work of Blundell et al. and outlined in section 3.1. Additionally, we have modified the loss to also include the KL divergence, as we seek to minimize this, and we have also updated the predictor to leverage the stochasticity of the model. With only these three modifications, we have converted a traditional U-Net to a Bayesian U-Net that gives uncertainty masks with each prediction mask!

The implementation of the Bayesian CNN was adapted from the work of Shridhar et al. [30], who give both a direct PyTorch implementation of the method of Blundell et al. and a version with a local reparameterization technique proposed by Kingma [19] that offers added efficiency. They are drop-in replacements Conv2d and linear layers. In appendix A.2, we consider the forward pass of the normal BNN convolutional layer.

In addition to the layer modification, we update the loss function to include the KL loss. The new loss function is the sum of the Cross Entropy, Dice Loss, and KL Divergence divided by the length of the training set, 6200 in our case.

Finally, we calculate uncertainty per pixel using the slight modification to the formulation of Kwon et al. [20]. We have rescaled their metrics to maintain the same units as the softmax outputs by taking the square root; on further consideration, this might not be necessary for future experiments. We take T samples of the model. That is, we pass the input through the model T times, each time getting a random prediction. Recall that we now have random weights that are defined by normal distributions with set mean and variance. The final prediction is the mean of the T samples, and the uncertainties are calculated by:

$$u_k^{aleatoric}(x) = \sqrt{\left[\frac{1}{T} \sum_{t=1}^T \text{diag}(\hat{p}_t) - \hat{p}_t^{\otimes 2} \right]_{kk}} \quad (6.3)$$

$$u_k^{epistemic}(x) = \sqrt{\left[\frac{1}{T} \sum_{t=1}^T (\hat{p}_t - \bar{p})^{\otimes 2} \right]_{kk}} \quad (6.4)$$

for class k and input pixel x , where $\bar{p} = \frac{1}{T} \sum_{t=1}^T \hat{p}_t$. \hat{p}_t is the vector of softmax outputs of the BNN that have been passed through the final Softmax function and $v^{\otimes 2} = vv^T$. The weights of the BNN are sampled each time randomly from their respective distribution.

This is done per pixel, and per class. The final mask per class can be averaged to get a single uncertainty mask for both aleatoric and epistemic varieties. Alternatively, we can look at the uncertainties for each class separately. Here, we look at the mean uncertainty across all the classes. We then average all the pixels in each mask, and produce a single metric per image for both kinds of uncertainty.

We present and analyze some sample results here to show efficacy.

6.4 Related Work

Late into our research and during our experiments, a similar work was published with some key differences. Dechesne et al. [8] propose a Bayesian U-Net architecture that, instead of using Bayes by Backprop, trains a random model through Monte Carlo dropout. In this approach, the dropout layers are left on even during prediction, thus providing a random model similar to the one we

train with the exception that instead of approximating with Gaussian distributions, weights are approximated with Bernoulli distributions. This more simple approach has an efficiency advantage but limits the complexity that can be learned through model training. Further, here it appears that the Bernoulli probability is manually set during training, instead of learned.

Instead of providing epistemic and aleatoric uncertainties directly, Dechesne’s work provides predictive entropy and mutual information, with the latter being “mutual information between the predictive distribution and the posterior over network weight.” [8]

Dechesne’s experiments were conducted on satellite imagery which generally has a GSD (ground surface distance) of 50cm or higher. Our experiments are conducted on drone imagery with a GSD of approximately 8cm. This lower GSD provides for data with much more detail and provides for qualitatively different results. Further, we believe that applying the the Bayes by Backprop approach described by Blundell et al. [4] directly and allowing the weights to be modeled with two-parameter Gaussian distributions provides a model with a greater richness of uncertainty information.

Chapter 7

Results on Aerial Segmentation

7.1 Initial Tests

In order to test the new model architecture, we first consider results in the absence of any specific tuning of the model or adjustment of hyperparameters. Our goal here is simply to show the efficacy of our uncertainty estimators in determining the validity of a prediction. As this dataset suffers from classes with low representation, we see poorer predictions on such classes. However, these initial tests already demonstrate strong potential in the performance of the uncertainty predictors.

In the results shown here, we are predicting on the entire dataset. In our full predictions presented later in Section 7.2, we will separate our data into train, test, and validation sets. Further, since our loss functions are different (the Bayesian U-Net also seeks to minimize KL divergence of the weights), we get varying behavior of predictions that is most apparent in classes of low representation. A metric that is more sensitive to class imbalance might improve this problem for future training runs.

We present some results with uncertainty values that have been linearly scaled from $[0, 1]$ based on the full set of results. This is done for greater ease of visualization and interpretation of the uncertainty values.

In figure 7.1, we see a disc of concrete and structure that is annotated as clutter. Though both the U-Net and the Bayesian U-Net fail to predict it as clutter, the Bayesian U-Net shows high epistemic uncertainty there. This is due to the fact that the class “clutter” has only 2% representation in the dataset — the model simply has not seen enough examples of such data.

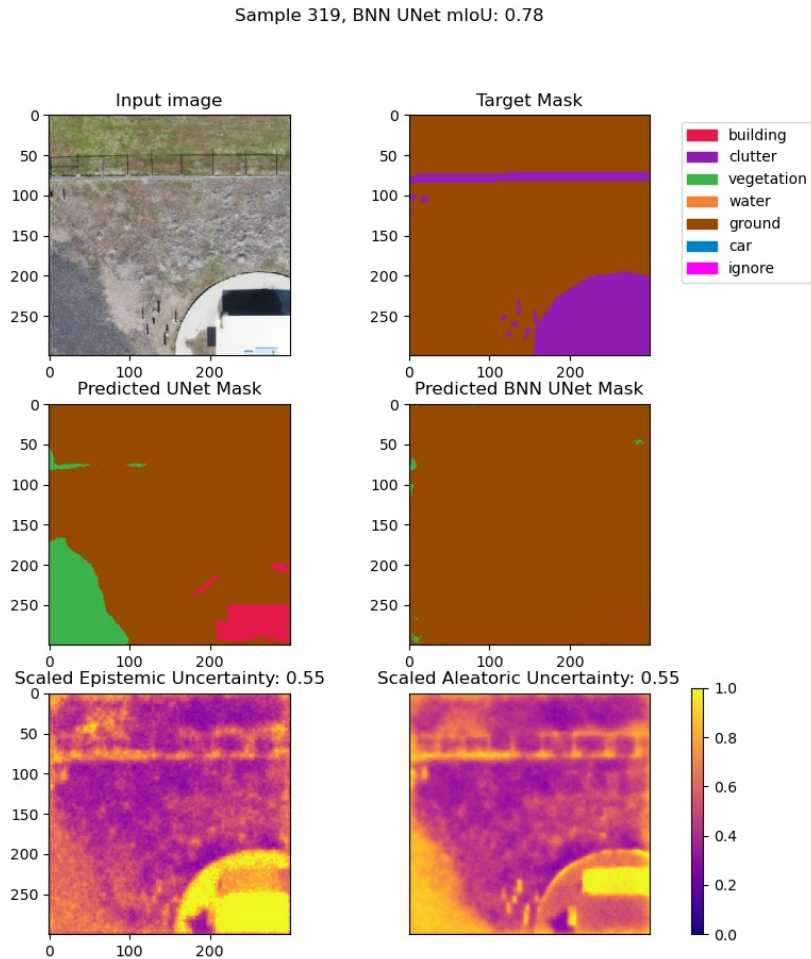


Figure 7.1: The disk area labeled as clutter shows high epistemic uncertainty. This disk is not predicted correctly by either the U-Net or the Bayesian U-Net.

In contrast, figure 7.2 shows the area of clutter with high aleatoric uncertainty. The model is still telling us it is uncertain here, but more so because the contents of the “clutter” look similar to other classes it has seen. This area has a lot of aleatoric noise.

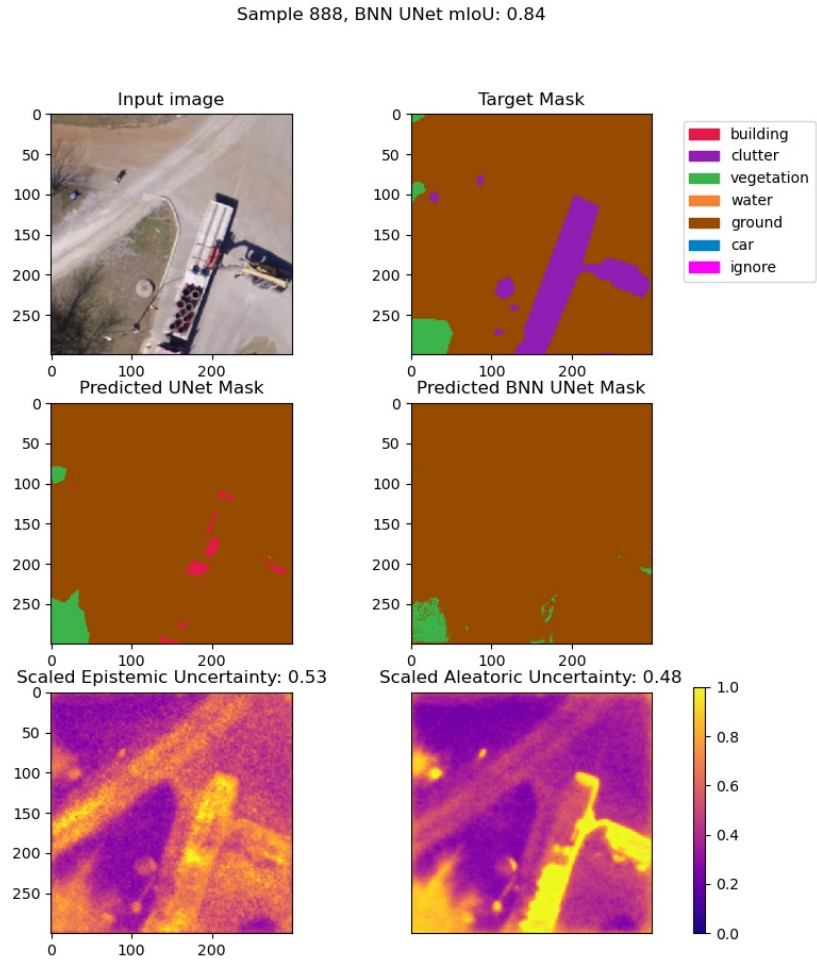


Figure 7.2: The area annotated as “clutter” shows high aleatoric uncertainty.

Figure 7.3 shows an example where the Bayesian U-Net correctly predicts the whole scene, all of which is ground, and does so with high confidence.

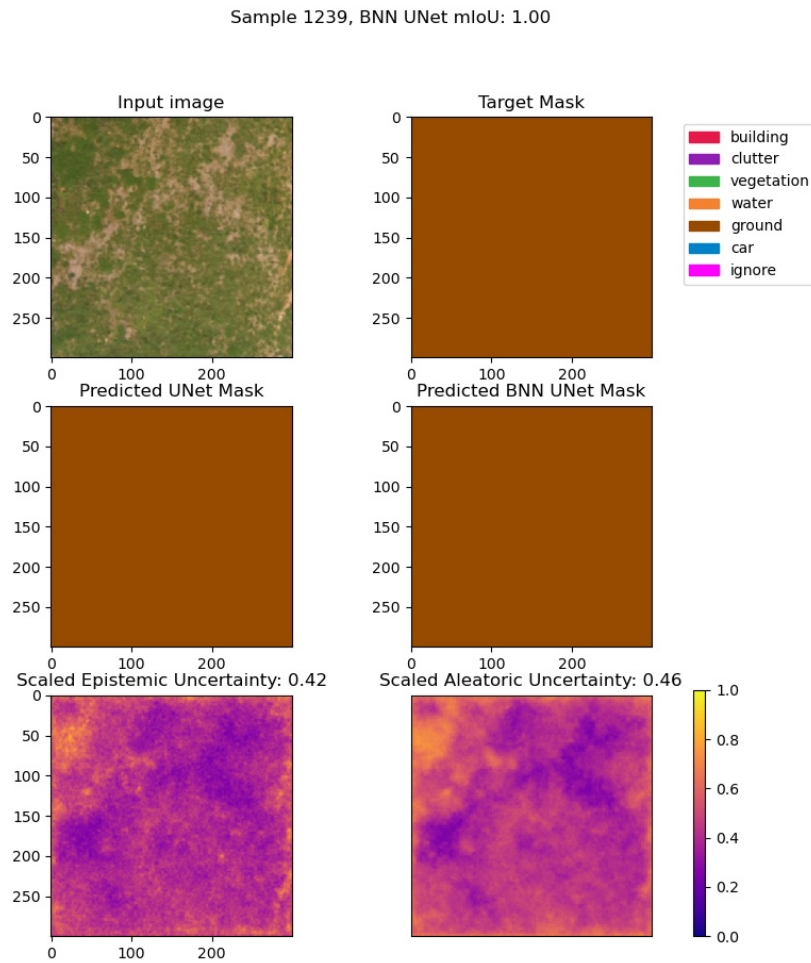


Figure 7.3: This sample is unambiguously ground to each model, and shows very low uncertainties.

The Bayesian U-Net does not predict buildings as well as the normal U-Net, while the normal U-Net tends to over-predict the class “building”. This is due to differences in the loss function. We do note, however, that they achieve a near-identical mIoU. In figure 7.4, we see the Bayesian U-Net failing to predict the building as building, but showing a high epistemic uncertainty in this area, thus telling us the prediction has low confidence.

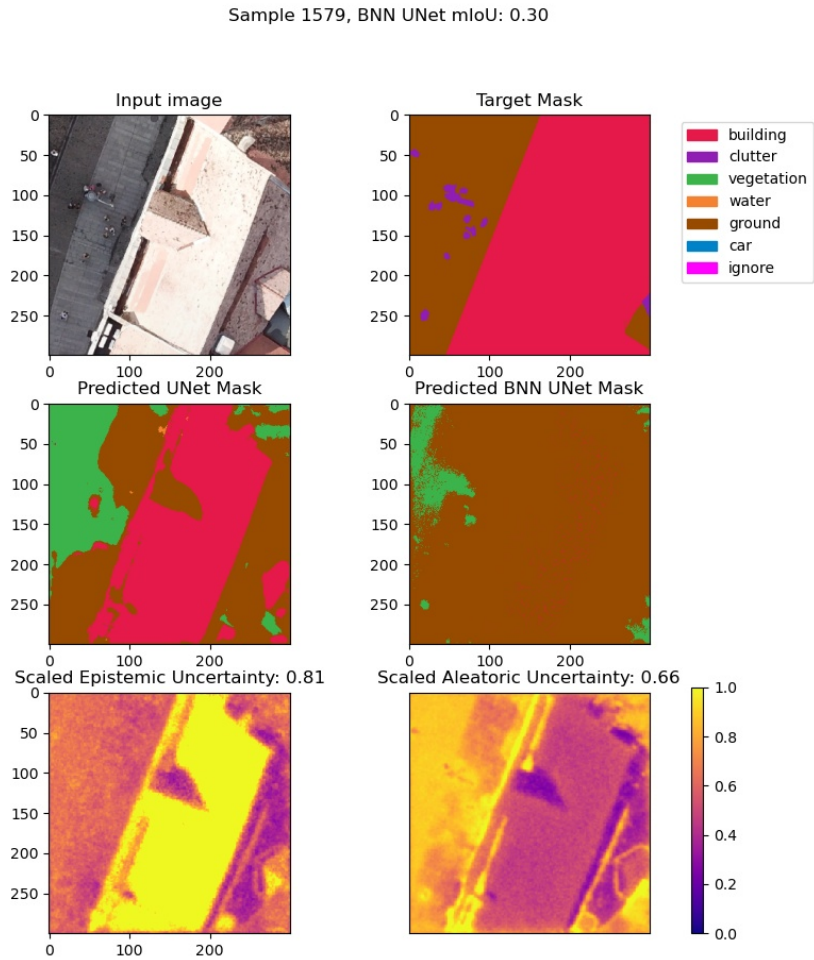


Figure 7.4: While the Bayesian U-Net does not predict the building well, it shows high epistemic uncertainty where the building is.

In our dataset, snow is annotated as ground, because it is not present in most of the data. We see in figure 7.5 that the normal U-net incorrectly identifies some areas of snow as building, while the Bayesian U-Net correctly identifies snow as ground. Because of the low class representation, we see a high amount of epistemic uncertainty where there is snow.

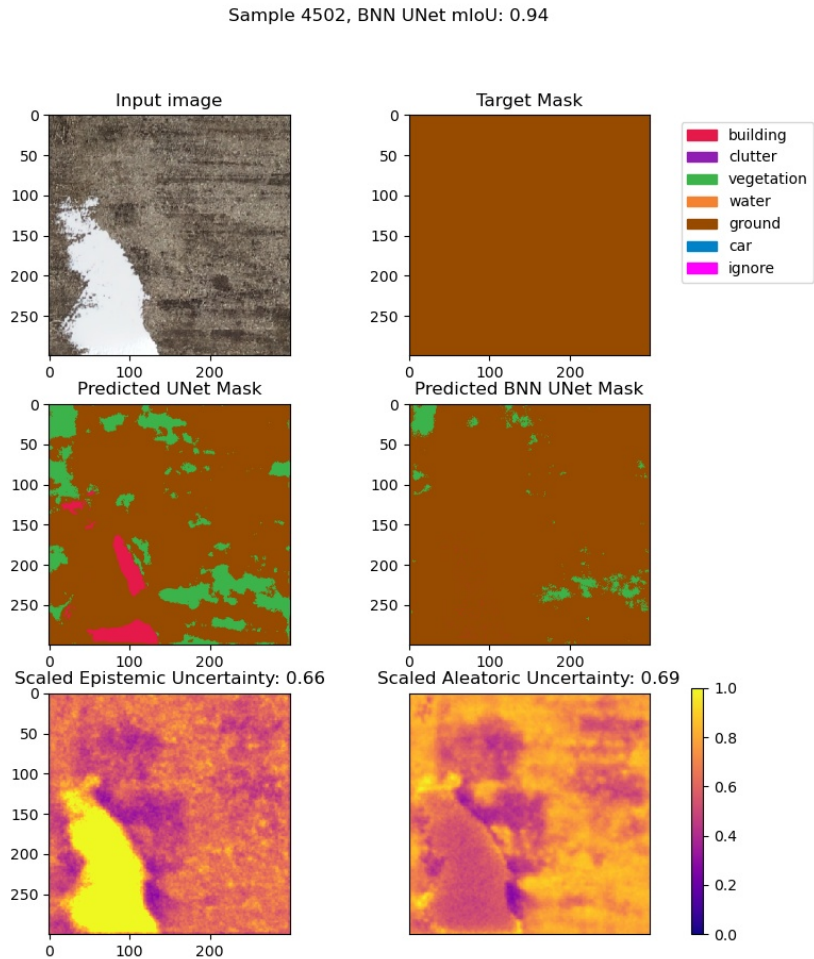


Figure 7.5: The presence of snow in the data is very small, thus we see high epistemic uncertainty from the Bayesian U-Net.

Finally, we look at an example where there is an ambiguous border between ground and vegetation. Both models struggle to identify the border, and the standard U-Net incorrectly identifies parts of the area as building. This ambiguity is defined as aleatoric noise and is evident in our Bayesian models' identification of such noise, as seen in figure 7.6.

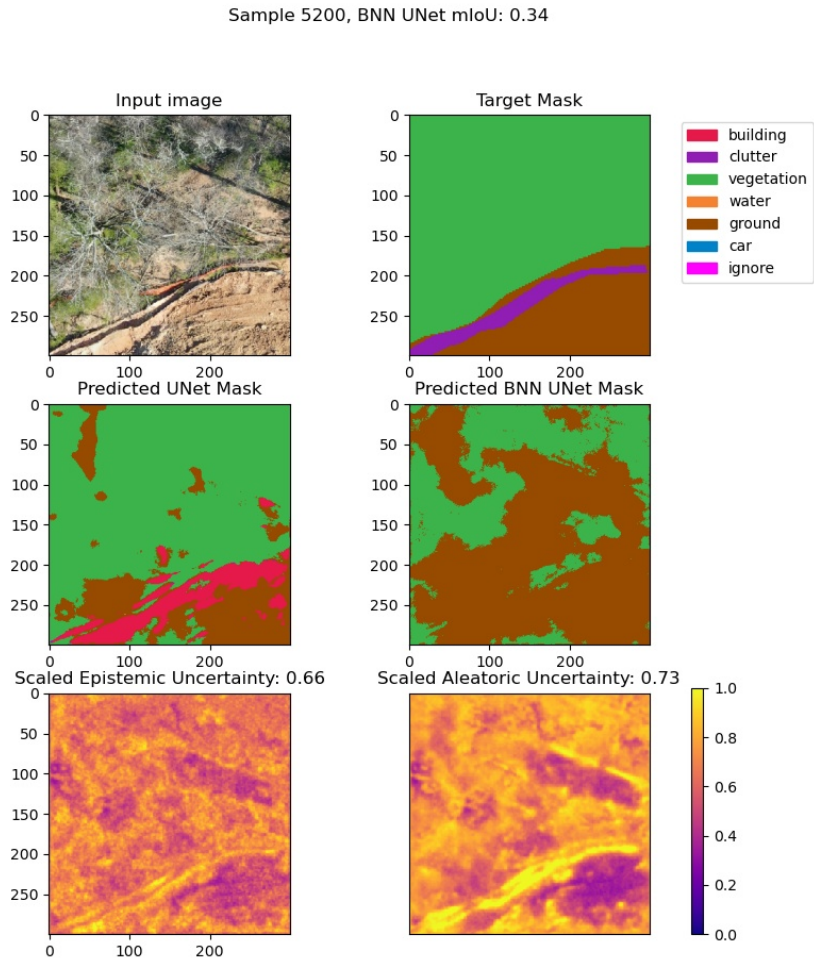


Figure 7.6: The division between ground and vegetation is ambiguous to the model, and this results in high aleatoric uncertainty.

7.2 Full Predictions

We then moved from an overfit preliminary test to a full experiment. The full dataset was split into train, test, and validation sets. Of 6888 annotated image chips, 10% were held out for final validation, and 10% of the remaining were used in model testing during the training process.

The resulting model was then run on the unseen validation data. The prediction mask, epistemic mask, and aleatoric mask; along with mIoU, mean epistemic uncertainty, and mean aleatoric uncertainty were recorded. Both epistemic and aleatoric uncertainties have been multiplied by a constant such that their average value is approximately 0.5. To rescale, we looked at a sample of predictions on the *training* data, and took the average across all epistemic and aleatoric uncertainties. Thus, the rescaling factor is set to be constant across all predictions. This rescaling is not necessary; but, in order to effectively use the estimates, some sort of thresholding must be applied and the uncertainties must be considered with respect to the distribution of all of the uncertainties.

The benefit of this scaling is that the uncertainty masks that are produced and visualized now have a meaningful range that is approximately $[0, 1]$, where 0.5 is the mean uncertainty. It is important to note, however, that a scaled uncertainty of 0.5 does not directly equate to 50% certainty. Instead, the value should be thresholded and considered in relation to the rest of the data.

From the validation data, the correlation coefficient between the epistemic uncertainty and the mIoU was measured to be -0.44 (see figure 7.7) and the correlation coefficient between the aleatoric uncertainty and the mIoU was measured to be -0.55 . This shows a negative correlation between the uncertainty and the mIoU. For greater uncertainty, the expected mIoU goes down, and vice versa. Should we have chosen a raw threshold of 0.052 aleatoric Uncertainty and kept only predictions below this, we would have an mIoU 0.87 with 20% of the validation data remaining. This is a very encouraging result in the efficacy in using this method in a production environment to understand the quality of the models predictions and to know when a model is starting to “go stale” (no longer producing good predictions on the stream of incoming data).

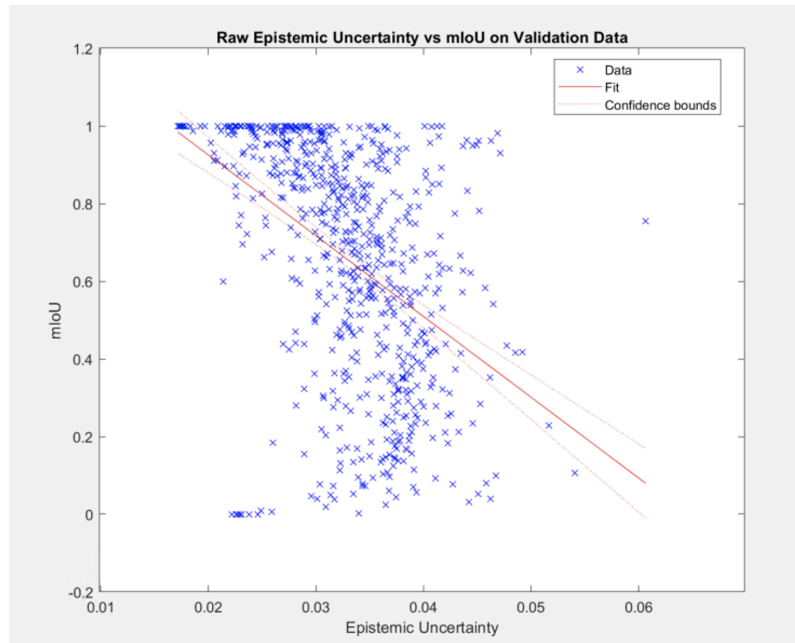


Figure 7.7: We see a negative correlation between the raw epistemic Uncertainty and the mIoU, suggesting that the metric can be used to gauge prediction quality.

We present some results from the full experiment. In figure 7.8 we see low uncertainty values for the ground class for which the Bayesian U-Net predicts with perfect mIoU. The base U-Net model did not predict most of the image properly; this, again, is due to variation in the loss function and the low representation of the “building” class, and should not be taken to mean that the Bayesian U-Net gets a higher mIoU by default.

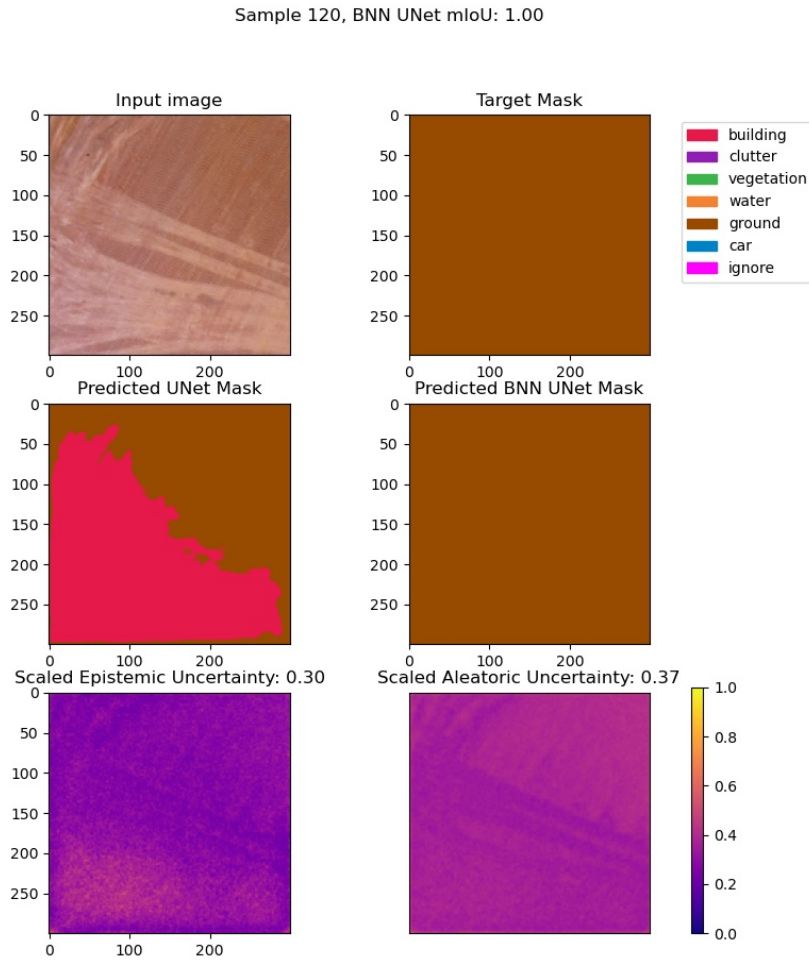


Figure 7.8: Unseen validation data: The ground class is of low uncertainty and correctly predicted by the Bayesian U-Net.

In figure 7.9, we see an input image with snow that appears white, but not fully washed out. The Bayesian U-Net shows high epistemic uncertainty here, indicating that we do not have many examples of snow, but it still predicts the class correctly (while the base U-Net fails).

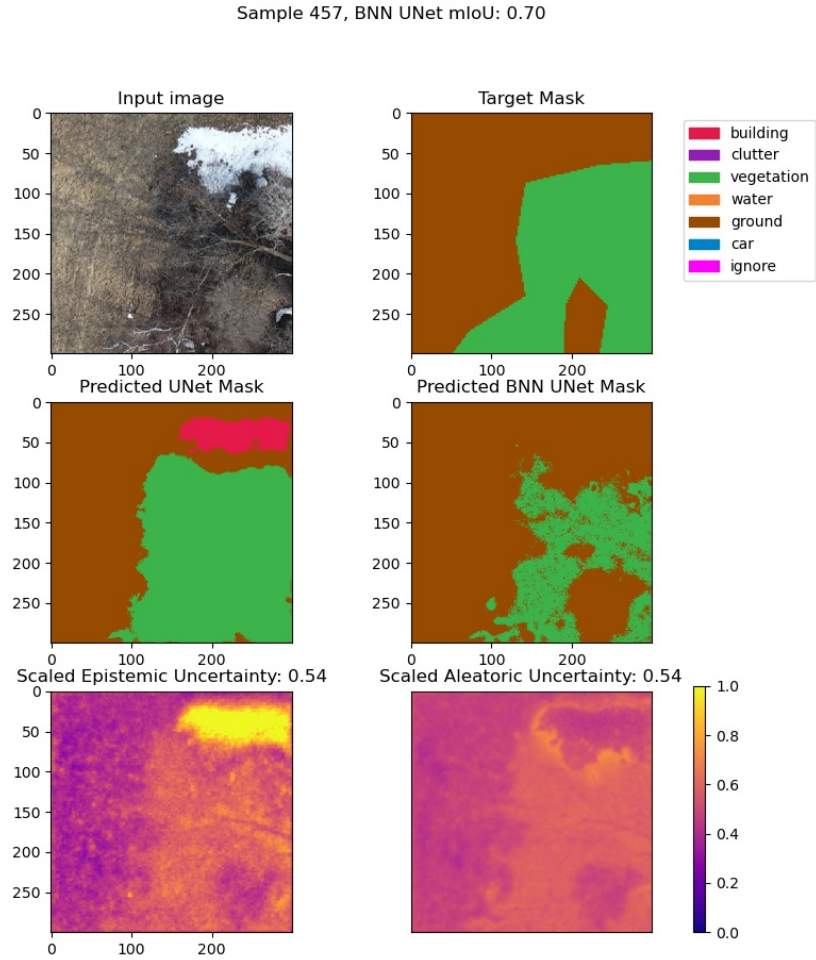


Figure 7.9: Unseen validation data: White patches and snow show high epistemic uncertainty.

In figure 7.10 we see a white truck with target class "clutter" that has high epistemic uncertainty and is incorrectly predicted as building from the base U-Net and ground from the Bayesian U-Net. One might expect that, with more data and better class labels, the model would learn from contextual information how to correctly classify vehicles on the road.

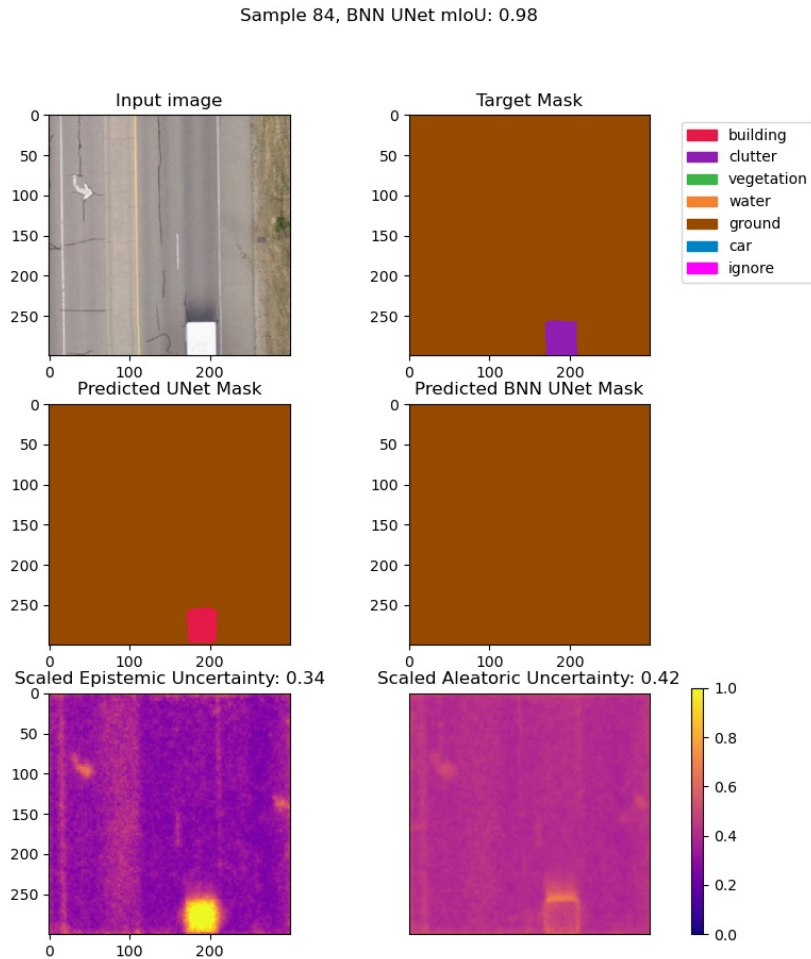


Figure 7.10: Unseen validation data: The washed out truck is misclassified and shows high epistemic uncertainty.

In validation figure 7.11 below, we again see an ambiguous forest with shadows, snow, and barren trees. The Bayesian U-Net seems to capture this ambiguity more in the output than the base U-Net, showing splotches of vegetation and ground, while the target class is mostly vegetation. The uncertainties are relatively high across the whole image.

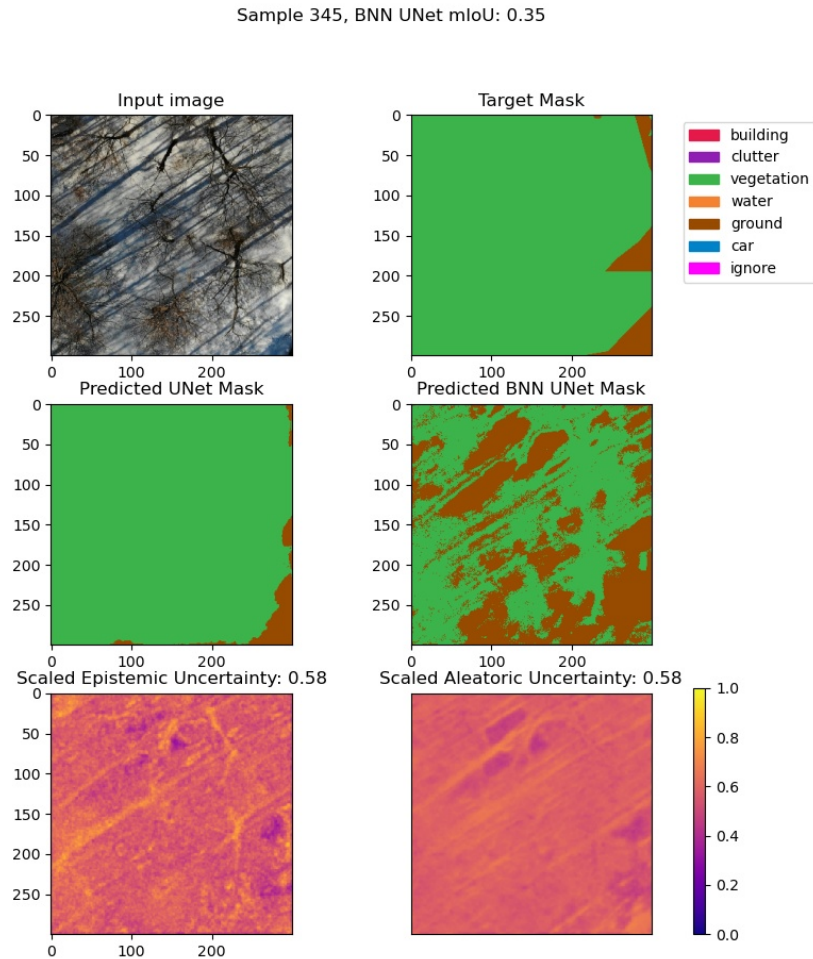


Figure 7.11: Unseen validation data: The division between vegetative forest and ground is still ambiguous to the model and it shows above average uncertainty.

7.3 Varying Data Size

A reduction in the amount of data should affect the uncertainty. Recall that epistemic uncertainty is that which arises from what the model does not know, and aleatoric uncertainty results from noise

in the data. Therefore, we would expect a random reduction in the training data (while keeping the validation dataset constant) to increase epistemic uncertainty and leave aleatoric uncertainty with little difference.

We have randomly reduced our training dataset to 70% and 50% sizes, and have compared the mean, raw epistemic and aleatoric uncertainties in table 7.1.

Data Size	Mean Epistemic	Mean Aleatoric
100%	0.0331	0.206
70%	0.0384	0.218
50%	0.0382	0.222

Table 7.1: Comparison of epistemic and aleatoric uncertainty for varying dataset size.

As expected, the epistemic uncertainty increases with a 70% dataset size compared to the original, though does not change much when further reducing to a 50% size. The aleatoric uncertainty, on the other hand, only increases marginally while reducing the dataset size. This gives good indication that each uncertainty metric is statistically performing as expected.

Since the metric rescaling was performed based on the metrics on the validation dataset, it does not make sense to rescale for this comparison. Accordingly, we might want to consider alternative rescaling schemes in the future.

Chapter 8

Conclusions and Further Perspectives

8.1 Summary

In this work, we started with an introduction to uncertainty metrics and divided them into epistemic and heteroscedastic aleatoric counterparts. We have tried to develop an intuitive understanding of each of these components of uncertainty. We then surveyed the leading methods of determining each.

We have experimented with an autoencoder as a standalone model for epistemic uncertainty estimation, and tested with MNIST and non-MNIST datasets.

Then, we introduced the Bayesian neural network and its construction. We considered its connection to variational autoencoders. We have implemented a one-layer BNN to a regression problem, and used the stochasticity of the BNN to construct a confidence interval of the model output. We have also explored an analytical approach to characterizing a dataset, though this approach appears intractable for complex networks. Additionally, we have studied a split-head BNN model for our regression problem.

Our main contribution involves a Bayesian U-Net for aerial segmentation. With our novel construction, we have trained on the open DroneDeploy dataset, and measured both per-pixel and image

wide uncertainty on our validation dataset. In addition to individual examples, we have shown the negative correlation of uncertainty to model performance as measured by the mIoU metric. Furthermore, we have run our Bayesian U-Net on datasets of randomly reduced size while keeping the validation set constant. This confirms a general increase in epistemic uncertainty when the data size is reduced.

8.2 A Workflow for Improving Model Performance

We propose a workflow for iterative model improvement that leverages each uncertainty metric covered, and is the topic for further research. This workflow is drawn out in figure 8.1.

Machine learning model performance can be thought of as an inversely correlated function to the models uncertainties. Structural uncertainty, or that which arises from the choice of model and hyper-parameters, is often that of most interest. New models and hyper-parameter training offers sometimes marginal gains on performance.

Understanding the effect of data on model performance is often limited to the understanding that more data leads to better performance. But what is not often considered is that when aleatoric uncertainty is present in the data, more data with the same amount of aleatoric uncertainty will not fix the problem. Further, when there is epistemic uncertainty present in the data that can

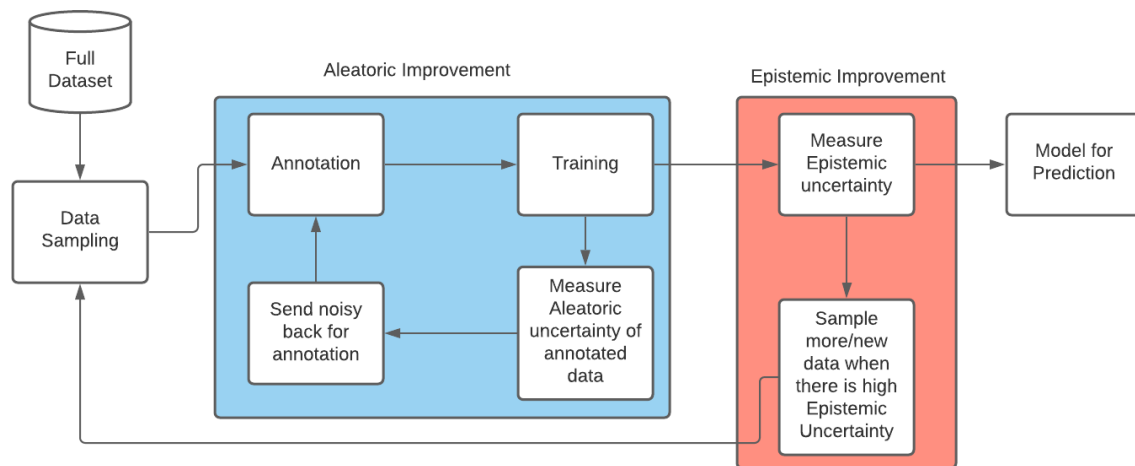


Figure 8.1: Proposed workflow for continuous model performance improvement.

be improved by more annotation samples, it is often not known where to draw these samples from.

By appropriately considering metrics for both epistemic and aleatoric uncertainty, we can improve the model training process that is aimed at achieving a specific accuracy metric. It is important to address the cause of each of these uncertainties separately.

High aleatoric uncertainty can arise from either poor annotation quality or ambiguous annotations. After a round of training on a dataset, one can then predict on the entire annotated dataset using bootstrapping. That is, we predict on the very data we just trained on, and measure the uncertainties of each.

By sorting based on aleatoric uncertainty (from an autoencoder or BNN), we can identify the data with highest aleatoric noise. In the case of poor annotation quality, we can then remove a portion of the noisy data. If there is a discrepancy in the annotations, we may then go back and correct these annotations. This gives a method for automatically checking the quality of the whole annotation batch and highlighting the noisiest of the annotations.

Epistemic uncertainty can be the effect of low representation, but can also result from predicting on new data that has not yet been seen by the model. For this reason, epistemic uncertainty must be monitored as new data arrives. When the epistemic uncertainty exceeds a threshold, for some buffer size, this is a clear indication that more of the new data samples must be annotated. For any system with an inflow of data that is being predicted on; we must consider a continuous annotation process if we want to maintain the same prediction quality. As epistemic uncertainty increases, we take these samples of high epistemic uncertainty and annotate them, then retrain the model. By annotating on a mixture of samples that have high uncertainty, and some that are randomly chosen, we may boost the model performance on the new data.

Appendix A

Code Excerpts

A.1 U-Net Pytorch Code Excerpts

Our implementation of the U-Net architecture in PyTorch [25] is defined by (where “bilinear” is a flag for the upsampling interpolation method):

```
self.inc = DoubleConv(n_channels, 64)
self.down1 = Down(64, 128)
self.down2 = Down(128, 256)
self.down3 = Down(256, 512)
factor = 2 if bilinear else 1
self.down4 = Down(512, 1024 // factor)
self.up1 = Up(1024, 512 // factor, bilinear)
self.up2 = Up(512, 256 // factor, bilinear)
self.up3 = Up(256, 128 // factor, bilinear)
self.up4 = Up(128, 64, bilinear)
self.outc = OutConv(64, n_classes)
```

where the Down method is defined by:

```
nn.MaxPool2d(2),
DoubleConv(in_channels, out_channels)
```

and the Up method by:

```
if bilinear:
    self.up = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
    self.conv = DoubleConv(in_channels, out_channels, in_channels // 2)
else:
    self.up = nn.ConvTranspose2d(in_channels, in_channels // 2, kernel_size=2, stride=2)
    self.conv = DoubleConv(in_channels, out_channels)
```

DoubleConv is defined by:

```
nn.Conv2d(in_channels, mid_channels, kernel_size=3, padding=1),
nn.BatchNorm2d(mid_channels),
nn.ReLU(inplace=True),
nn.Conv2d(mid_channels, out_channels, kernel_size=3, padding=1),
nn.BatchNorm2d(out_channels),
nn.ReLU(inplace=True)
```

The loss function is defined as the sum of the Cross Entropy and the Dice Loss, with an implementation that looks like:

```
loss = nn.CrossEntropyLoss(masks_pred, true_masks) \
    + dice_loss(F.softmax(masks_pred, dim=1).float(),
                F.one_hot(true_masks, net.n_classes).permute(0, 3, 1, 2).float(),
                multiclass=True)
```

A.2 Bayesian U-Net Pytorch Code Excerpts

To better understand the Bayesian modification to the PyTorch U-Net implementation, we take a look at the forward pass of the Bayesian convolutional layer.

```
W_eps = torch.empty(self.W_mu.size()).normal_(0, 1).to(self.device)
self.W_sigma = torch.log1p(torch.exp(self.W_rho))
weight = self.W_mu + W_eps * self.W_sigma
```

This is the exact implementation as described by Blundell et al. [4] and detailed in section 3.3.

$$\epsilon \sim \mathcal{N}(0, I) \tag{A.1}$$

$$\mathbf{w} = \mu + \log(1 + \exp(\rho)) \cdot \epsilon \tag{A.2}$$

The BNN convolutional layer with local reparameterization does this a little more efficiently, where the mean and variance are precomputed with convolutional filters over the data.

```
act_mu = F.conv2d(
    x, self.W_mu, self.bias_mu, self.stride, self.padding, self.dilation, self.groups)
act_var = 1e-16 + F.conv2d(
    x ** 2, self.W_sigma ** 2, bias_var, self.stride, self.padding, self.dilation,
    self.groups)
act_std = torch.sqrt(act_var)

eps = torch.empty(act_mu.size()).normal_(0, 1).to(self.device)
return act_mu + act_std * eps
```

Bibliography

- [1] Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2(1):1–18, 2015.
- [2] Roman V Belavkin. Relation between the Kantorovich–Wasserstein metric and the Kullback–Leibler divergence. In *Information Geometry and its Applications IV*, pages 363–373. Springer, 2016.
- [3] Boris Belousov. Geodesic distance between probability distributions is not the KL divergence, Jul 2017. URL: <http://boris-belousov.net/2017/07/11/distance-between-probabilities/>.
- [4] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622. PMLR, 2015.
- [5] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848, 2017.
- [6] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.
- [7] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. EMNIST: Extending MNIST to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926. IEEE, 2017.

- [8] Clément Dechesne, Pierre Lassalle, and Sébastien Lefèvre. Bayesian U-Net: Estimating uncertainty in semantic segmentation of earth observation images. *Remote Sensing*, 13(19):3836, 2021.
- [9] DroneDeploy. DroneDeploy machine learning segmentation benchmark, Oct 2019. URL: <https://github.com/dronedeploy/dd-ml-segmentation-benchmark>.
- [10] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.
- [11] Andrei Dmitri Gavrilov, Alex Jordache, Maya Vasdani, and Jack Deng. Preventing model overfitting and underfitting in convolutional neural networks. *International Journal of Software Science and Computational Intelligence (IJSSCI)*, 10(4):19–28, 2018.
- [12] Alex Graves. Practical variational inference for neural networks. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.
- [13] Aditya Grover and Stefano Ermon. Uncertainty autoencoders: Learning compressed representations via variational information maximization. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2514–2524. PMLR, 2019.
- [14] Michael R Heffels and Joaquin Vanschoren. Aerial imagery pixel-level segmentation. *arXiv preprint arXiv:2012.02024*, 2020.
- [15] Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, pages 5–13, 1993.
- [16] Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: A tutorial introduction. *arXiv preprint arXiv:1910.09457*, 2019.
- [17] Michel Kana. Uncertainty in deep learning. How to measure?, May 2020. URL: <https://tinyurl.com/2p99cm8s>.

- [18] Alex Kendall and Yarin Gal. What uncertainties do we need in Bayesian deep learning for computer vision? *arXiv preprint arXiv:1703.04977*, 2017.
- [19] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.
- [20] Yongchan Kwon, Joong-Ho Won, Beom Joon Kim, and Myunghee Cho Paik. Uncertainty quantification using Bayesian neural networks in classification: Application to ischemic stroke lesion segmentation. 2018.
- [21] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [22] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.
- [23] Han Liu, Zipeng Fu, Yipeng Li, Nazreen Farina Ahmad Sabri, and Mathieu Bauchy. Balance between accuracy and simplicity in empirical forcefields for glass modeling: insights from machine learning. *Journal of Non-Crystalline Solids*, 515:133–142, 2019.
- [24] Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. Adversarial variational Bayes: Unifying variational autoencoders and generative adversarial networks. In *International Conference on Machine Learning*, pages 2391–2400. PMLR, 2017.
- [25] Alexandre Milesi. Milesial/pytorch-unet: Pytorch implementation of the U-Net for image semantic segmentation with high quality images, 2021. URL: <https://github.com/milesial/Pytorch-UNet>.
- [26] Joseph Rocca. Understanding variational autoencoders (VAEs), Mar 2021. URL: <https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73>.
- [27] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing*

- and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.
- [28] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.
- [29] Claude Elwood Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.
- [30] Kumar Shridhar, Felix Laumann, and Marcus Liwicki. Uncertainty estimations by softplus normalization in Bayesian convolutional neural networks with variational inference. *arXiv preprint arXiv:1806.05978*, 2018.
- [31] Kumar Shridhar, Felix Laumann, and Marcus Liwicki. A comprehensive guide to Bayesian convolutional neural network with variational inference. *arXiv preprint arXiv:1901.02731*, 2019.
- [32] Niclas Ståhl, Göran Falkman, Alexander Karlsson, and Gunnar Mathiason. Evaluation of uncertainty quantification in deep learning. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 556–568. Springer, 2020.
- [33] Panqu Wang, Pengfei Chen, Ye Yuan, Ding Liu, Zehua Huang, Xiaodi Hou, and Garrison Cottrell. Understanding convolution for semantic segmentation. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1451–1460. IEEE, 2018.
- [34] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.
- [35] Matthew D Zeiler, Graham W Taylor, and Rob Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *2011 International Conference on Computer Vision*, pages 2018–2025. IEEE, 2011.
- [36] Ruonan Zhang, Long Yu, Shengwei Tian, and Yalong Lv. Unsupervised remote sensing image segmentation based on a dual autoencoder. *Journal of Applied Remote Sensing*, 13(3):038501, 2019.