Uncertainty and Bayesian Neural Networks in Environmental Sound Classification

***Author*: Lorcan Donnellan**

01818119

***Supervisor*: Dr. Kevin Webster**

Thesis presented for the degree of Applied Mathematics MSc

Department of Mathematics

Imperial College London

October, 2021

Abstract

Regular neural networks, while being extremely powerful, are often prone to overfitting and are incapable of accurately calculating uncertainty in data. This can result in simultaneously overconfident and incorrect predictions. Coupled with the "black box" nature of neural networks, this can lead to a lack of trust in predictions. These drawbacks inhibit the use of the technology in mission critical tasks, examples of which are found in the medical industry, the developing industry of self-driving cars, and in military operations. A level of explainability would be useful in such cases. A potential solution to these problems is the Bayesian Neural Network (BNN). The BNN is a probabilistic variation on the regular neural network that allows us to calculate measures of certain types of uncertainty in predictions. This provides a framework with which to assess predictions in more detail.

In this thesis we introduce BNNs and implement one for an environmental sound classification task. Two of the main datasets for such tasks are ESC-50 and Urban Sounds 8K. We consider known deep learning results on these datasets and select one with the aim of reproducing it with a BNN in the place of the regular neural network, keeping the general architecture of the BNN the same as the regular neural network. We analyze the different parameters of the BNN and their effect on performance. Using this analysis we train a BNN with the aim of replicating the performance of its deterministic counterpart. This method also makes possible direct comparisons between the two. We exhibit the enriched results and uncertainty visualisations made possible by using a BNN. We calculate uncertainty in the model and interpret the results. Finally we draw conclusions on the efficacy of the implemented BNN for the task, drawing conclusions on the advantages and disadvantages.

Signed Declaration

The work contained in this thesis is my own work unless otherwise stated.

Signed: Lorcan Donnellan.

Dedications

To Rita Cunningham and the late Margaret Grindel, whom I can’t thank enough for their lasting inspiration. Thank you.

Acknowledgements

I would like to express my sincerest gratitude to my supervisor Dr. Kevin Webster for going above and beyond in his guidance and support throughout a difficult year. I would also like to thank Charlene, Martin and Ethan McGuire for providing a home away from home here in London. Finally, I would like to thank my parents and siblings for their continued support and encouragement.

Contents

Introduction

The use of audio data to inform decisions in today's society is becoming increasingly prevalent. From Spotify's use of CNNs in their recommendation system[42], to audio surveillance systems[66][53], to the array of use cases in modern smart cities[2], it's clear that the modern age provides many opportunities to use audio data daily in a positive way. Other interesting audio classification use cases are found in the health care industry (both medical[23] and administrative[84]) and wildlife preservation[51][88]. With so many opportunities to gather audio data, computationally effective and accurate methods for fulfilling such tasks are in demand. Machine learning methods provide good techniques with which to take advantage of these opportunities, as the references above testify to. In particular deep learning methods, which will be the focus of this paper, can be used to great effect in real life situations[19].

Much of audio classification and recognition research falls into the categories of speech recognition[75], music classification[20], and environmental sound classification[57], the latter of which will be the focus of this paper. There are tasks overlapping these categories or falling outside of these descriptions, like heart sound classification[23]. However, these three main disciplines have been extensively investigated and we can use them to draw comparisons on the different problems posed by each. Generally the sounds corresponding to the labels in environmental sound classification (ESC) tasks are heard amongst any number of other background environmental sounds. For instance, in the environmental sounds dataset Urban Sounds 8K[72], the file 101729-0-0-1.wav is labelled "air_conditioner" but crickets are arguably the most audible thing in the recording. This noise in the data is literal noise in the environment. This is a result of the microphone used to capture environmental sound data generally being located a distance away from the sound it captures. Many other sounds can be picked up in between the source of the labelled sound and the microphone. This can also result in a small signal-to-noise ratio in environmental sound data[21]. This is in contrast to some other audio classification tasks. For instance, in musical recordings microphones are placed near the sources and measures are usually taken to ensure as little background noise as possible is recorded. Sound engineers at venues are employed specifically to make these arrangements and microphone position has been shown to make a difference in quality of recordings[43]. Usually people using speech recognition devices speak into the microphone also. All in all, the unchoreographed nature of environmental sound data presents challenges different to other types of audio classification tasks.

For some of these use cases, whether or not a model predicts accurately on what it hears is of critical importance. For instance, in audio surveillance use cases of deep learning [66][53], it would be dangerous for a model to report a false negative on a potential criminal incident. Similarly when deep learning models are used in the medical industry[23][84]. In these scenarios it would be useful to have some measure of the uncertainty in the model so that if the model is uncertain, perhaps human intervention could be used or the prediction could be flagged in some way. When a human doesn't know the answer to a question, it knows it doesn't know and can research further or seek help. Regular deep learning models do not possess this attribute. When the stakes are high, measuring uncertainty is important. In this paper we would like to incorporate a measure of uncertainty into a known network which takes environmental sound data as its input. There are several ways to do this [34] but in this paper we will focus on measuring uncertainty through the use of Bayesian Neural Networks (BNNs). The basic premise is that instead of optimising point estimates of weight and bias values in a neural network, we replace these points with distributions and optimise the distribution parameters instead[11]. BNNs will be discussed and described in more detail in Section 1.

The rest of the thesis is outlined as follows. In the remainder of Section 1 we consider previous research in these topics and introduce the ESC dataset we will be using to train and test our BNN. In Section 2, we provide the mathematical theory underlying a BNN which includes a brief description of neural networks and convolutional neural networks in general. Following this is an implementation of a BNN in Section 3. This is the largest section in the thesis. We introduce the deterministic model we base our BNN architecture on. Having selected a result to replicate, we analyse five hyperparameters that showed interesting relationships to performance. We use the results of this analysis to see if we can achieve comparable results to that of the original neural network. Having settled on a BNN model, we can compare the BNN and the regular neural network. Finally, in Section 3, we exhibit the measures of uncertainty made possible by using a BNN and interpret the calculations of uncertainty. We finish the paper by drawing our conclusions in the final Section 4.

1.1 Prior Work

We research the existing literature on ESC and uncertainty in neural networks more generally. Much research has been done in this field, especially since the introduction of the two excellent datasets Urban Sounds 8k and ESC-50.

The first thing to note is the variety of possible features we can extract from audio data. Modelling of audio data is traditionally broken up into two parts. The first is designing a feature extraction to find a suitable representation of the audio data. The second is building and training a model to make predictions on this data. The acoustic features we extract from the data is very important and can have a great effect on the performance of a model, as is investigated by Zohaib Mushtaq and Shun-Feng Su, 2020[57]. There are many different options when it comes to this aspect. Audio data is generally stored as wav files, which can be read in plainly as a sequence of samples. These can be read in directly to give us the first and simplest feature we will mention: the raw wave form. These features preserve the time sequence of a audio track, which is lost when using some other features, like filterbank magnitude features. Raw wave forms have been used to good effect in deep learning frameworks[39]. Mel filter based features provide more options. These are based on the Mel scale, which is a scale of pitches that mimics human perception of sound. Mel frequency cepstral coefficients (MFCC), Mel, and Log-Mel spectrograms are amongst the features based on this scale. These features are visualisations of audio data, allowing for the full power of CNNs to be in this task (CNNs wre originally developed for computer vision). Successful results on ESC tasks can be achieved using these features and they are regularly used in research of sound classification in general[23][89][18]. Another group of features we mention is Gammatone filter-related features. These include the gammatone frequency cepstral coefficient based features, like the gammatone spectrogram. Gammatone frequency cepstral coefficients are are a biologically inspired modification on Mel frequency cepstral coefficients employing Gammatone filters with equivalent rectangular bandwidth bands. These have been shown to outperform log-mel spectograms[89][1]. These are a few of the main features we can extract from audio data. Feature extraction is an important component of audio classification process and comparisons of performance scores for different features on identical models referenced above testify to this.

We now go into how deep learning has been used in ESC tasks. Piczak, K.J.[64] applies machine learning methods (not falling under the category of deep learning) in an introduction to the ESC-50 dataset with reasonable results. The follow up paper[63] shows the promise of deep learning methods for this purpose. The latter paper also models the Urban Sounds 8k data, another widely used ESC dataset that we will describe in more detail later[72]. The paper shows a relatively simple model consisting of two convolutional layers and incorporating data augmentation that provides good results for both datasets. In the paper it is noted that these attempts are rudimentary but that the potential for more complex models is evident. It also shows the benefits of extracting image features from audio.

An excellent investigation into the effects of several data augmentation methods and image feature extractions is given in by Loris Nanni, Gianluca Maguolo, Sheryl Brahnam, Michelangelo Paci, 2021[59]. The paper combines each data augmentation method with a separate CNN and investigates the effect on performance. CNNs were pretrained on ImageNet and Places365 and fine tuned to the particular ESC-50 dataset thereafter. It also combines ensembles of neural networks to achieve state of the art results. It does provide some results for CNNs on ESC-50; the best stand-alone CNNs were VGG16 and VGG19 in this study with respective mean accuracies of 71.60% and 71.30% on the ESC-50 dataset (without augmentation).

Salamon and Bello[73] further make the case for data augmentation in ESC tasks, citing it as a way of overcoming the problem of data scarcity in the area. The technique is used on log-scaled mel-spectrograms, with the resulting inputs fed into a CNN comprised of 3 convolutional layers interleaved with 2 pooling operations, followed by 2 fully connected layers. Their proposed model with augmentation attains a state-of-the-art mean test accuracy of 79% on the Urban Sounds 8K dataset. Without data augmentation the model still performs favourably giving a mean test accuracy of 73%. The 79% test accuracy in particular is one of the best accuracies attained on the Urban Sounds 8K dataset without the use of ensembles. Other approaches using multi channel networks[44] and transfer learning models like Dense-161, with data enhancement techniques and multiple feature aggregation[57]. These are to name but a few state-of-the-art deep learning applications in environmental sound classification and on ESC-50 and UrbanSounds8K in particular. There are many more examples within the huge amount of research available in this area.

A large amount of analysis on the theory of Bayesian methods for data analysis has been contributed[35][8]. Jouko Lampinen and Aki Vehtari provide three use cases for Bayesian Neural networks[49]. They place a focus on the prior choice in particular, finding less restrictive priors yielded the best results. The use cases in this paper are a classification task in forest scene analysis, an inverse problem in electrical impedance tomography, and a regression task in concrete quality assumption. The first of these three tasks is the most relevant to us in this thesis as we will be carrying out a classification task also. The paper used a Bayesian multilayer perceptron for this classification task, and found that the Bayesian networks outperformed other non-Bayesian networks.

Other papers aim to bridge the gap between theory and real-life applications for Bayesian Neural networks. HH Thodberg uses a BNN for the real-life application from the meat industry of infrared spectroscopy[80]. More recent application include internet traffic classification[4], biomedical image segmentation[48], and ischemic stroke lesion segmentation[47]. These contributions show the promise of BNNs for classification tasks and how the enriched results can help us determine uncertainty. There are many ways to quantify uncertainty in predictions of a neural network but that provided by the use of BNNs has been found to be of the most reliable[34].

The references given above show how a BNN can be used effectively in a range of classification tasks to make predictions and quantify uncertainty. Despite this, no specific research can be found on applying such methods to environmental sound classification tasks, which is what we will be attempting in this paper.

1.2 Dataset: Urban Sounds

Urban Sounds provide two datasets, Urban Sounds and Urban Sounds 8K, and a taxonomy for urban sounds research. Compiled by J. Salamon, C. Jacoby and J. P. Bello[72], the datsets opened new doors for environmental sound research and classification, as datasets of this kind are generally difficult to arrange in a way that is amenable to training deep neural networks that generalise well to real world situations[30]. This is particularly true when compared to other classification datasets of music and speech. We will be using the Urban Sounds 8K dataset in the BNN implementation later.

Creating a taxonomy is often the first step in compiling a dataset for urban sound classification. The following three requirements of any taxonomy were put forward with Urban Sounds. The first was that it should consider previous taxonomies and progress made in and through their creation. Secondly, it should be as precise as possible, going down into as low a level of description as is feasible. Thirdly and finally, it should focus on sounds which are relevant in an urban setting, such as sounds that contribute to noise pollution. As noted in the description of the Urban Sounds taxonomy, there is no obvious consensus amongst the literature on how to construct a taxonomy. In order to satisfy the first requirement above, the Urban Sounds taxonomy is based on the work by Brown[14]. The taxonomy by Brown was built on previous proposals[67], and is one of the most detailed taxonomies for sounds in the urban acoustic environment. Four main groups common to most previous works were defined: human, nature, mechanical and music. To satisfy the second requirement, leaves of the graph go down into the lowest levels possible, so as to be as unambiguous as possible. The third requirement was satisfied by using noise complaints filed through New York City's 311 service from 2010-2014 in order to get a grasp on what were common sounds heard in an urban environment. Figure 1.1 shows the resulting taxonomy.

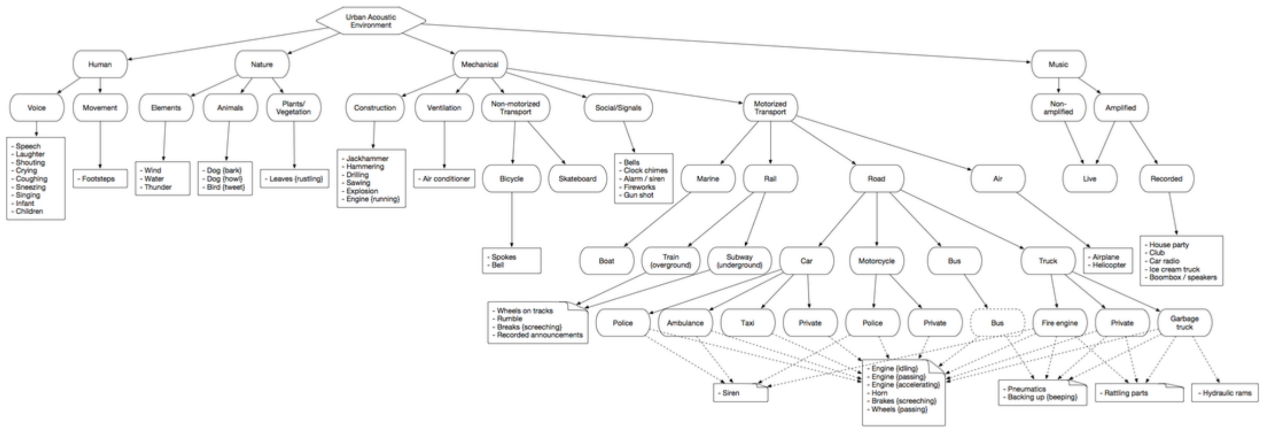

Urban Acoustic Environment
Human
Nature
Mechanical
Music
Voice
Movement
Elements
Animals
Plants/ Vegetation
Construction
Ventilation
Non-motorized Transport
Social/Signals
Motorized Transport
Non-amplified
Amplified
- Speech - Laughter - Shouting - Crying - Coughing - Sneezing - Singing - Infant - Children
- Footsteps
- Wind - Water - Thunder
- Dog (bark) - Dog (howl) - Bird (tweet)
- Leaves (rustling)
- Jackhammer - Hammering - Drilling - Sawing - Explosion - Engine (running)
- Air conditioner
Bicycle
Skateboard
- Bells - Clock chimes - Alarm / siren - Fireworks - Gun shot
Marine
Rail
Road
Air
Live
Recorded
- Spokes - Bell
Boat
Train (overground)
Subway (underground)
Car
Motorcycle
Bus
Truck
- Airplane - Helicopter
- House party - Club - Car radio - Ice cream truck - Boombox / speakers
- Wheels on tracks - Rumble - Breaks (screeching) - Recorded announcements
Police
Ambulance
Taxi
Private
Police
Private
Bus
Fire engine
Private
Garbage truck
- Siren
- Engine (idling) - Engine (passing) - Engine (accelerating) - Horn - Brakes (screeching) - Wheels (passing)
- Pneumatics - Backing up (beeping)
- Rattling parts
- Hydraulic rams

Figure 1.1: Urban Sounds Taxonomy

The Urban Sounds 8K dataset was compiled as a collection of annotated examples of ten lower level classes from the taxonomy. It was created for research on sound source identification and consists of a subset of short audio snippets of 4 seconds long. The classes were specifically chosen based on the frequency with which they appeared in the New York City 311 service noted above. The ten classes are air_conditioner, car_horn, children_playing, dog_bark, drilling, enginge_idling, gun_shot, jackhammer, siren, and street_music (as labelled in the Urban Sounds documentation[72]). Similarly to constructing the taxonomy, three main goals were set. Firstly, perhaps most obviously, only sounds that appear in an urban sound environment should actually appear. This makes for a better performing model in real life situations. Secondly, only authentic field recordings should be used. Finally, the dataset should be sufficiently large and varied so that models trained on it should generalise reasonably well to real world examples. Freesound provides access to 160,000 field recordings matching these requirements[83][30]. Recordings are user-uploaded and free to use for research. All recordings in the Urban Sounds datasets were taken from here. The Freesound database was searched (under specific class name based queries) and downloaded from using the Freesound API. Further manual checks were made and metadata assembled. A full account of the assembly of this dataset can be found in the paper[72]. In the end there are 8732 sound excerpts, all under 4 seconds. We will use this dataset to train and test the model we implement later.

1.2.1 Exploratory Data Analysis

Code used for this section can be found here[25], as developed by the author. All results are reproducible. We can load in the UrbanSounds8K.csv file in the metadata folder to see all the information given on each track. Here we have a table with columns slice_file_name, fsID, start, end, salience, fold, classID, and class. The first of these, slice_file_name, is the main identifier. The name takes the following format: [fsID]-[classID]-[occurrenceID]-[sliceID].wav. fsId is the Freesounds identifier. The classID indicates which class the recording belongs to. occurenceID is a numeric identifier to distinguish different occurrences of the sound within the original recording. slideID is a numeric identifier to distinguish different slices taken from the same occurrence. The start and end columns refer to the times in the freesound recordings that the slices were taken from. The salience column is subjectively decided. A value of 1 means the label sound is heard in the foreground, while 2 implies it is heard in the background. Finally the fold column indicates the fold the track belongs to.

As stated previously, the dataset Urban Sounds 8K consists of 8732 sound tracks split into 10 classes. The paper[72] places a limit of 1000 samples in each class. We find there are 1000 samples in every class except for classes 1, 6 and 8 which have 429, 374 and 929 samples respectively. That is, the number of samples per class is non-uniform. All tracks are dispersed into 10 folds.

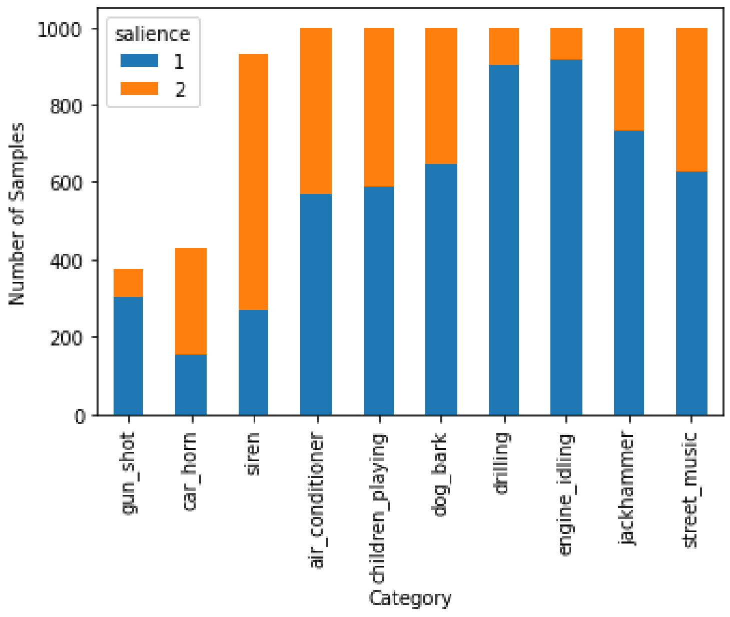
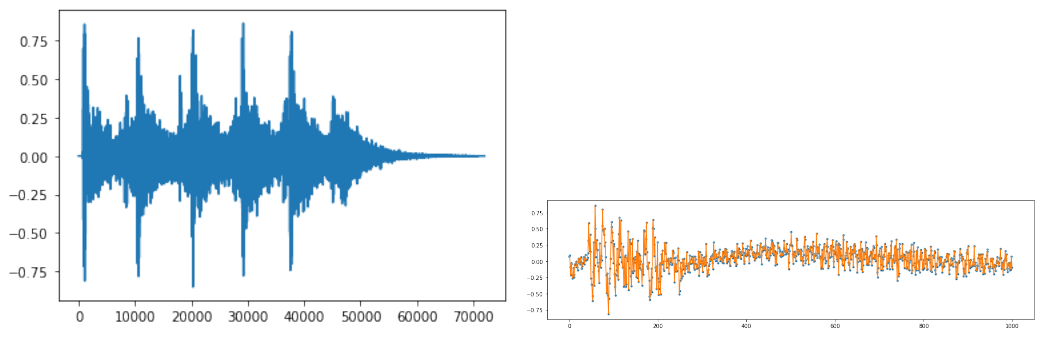


Figure 1.2: Urban Sounds 8K: Number of Audio Samples per Class

We can recreate the graph found in the paper[72]. We see here the breakdown of each class, how many audio tracks there are in each and how these are further broken down into two salience groups. We would expect examples with salience 1 to be easier to predict given that they are in the foreground. For this reason we can speculate that the classes gun_shot, jackhammer, drilling and engine idling will be amongst the better performing predictions. This turns out to be the case, as we see in our AUROC curves in Figure 3.13 where these classes have scores of over 94%. By comparison, the class siren scores the lowest of 73%. We see from the Figure 1.2 that it has the lowest proportion of salience 1 samples, suggesting the AUROC result is not surprising. The overall proportions of salience 1 to salience 2 in each class are given in the following table.

Class	**Salience Proportion**
gun_shot	4.342857
car_horn	0.554348
siren	0.407576
air_conditioner	1.320186
children_playing	1.427184
dog_bark	1.816901
drilling	9.204082
engine_idling	10.904762
jackhammer	2.717472
street_music	1.666667

We can load in the wav files and visualise them with plots. We load in a gunshot sound excerpt from fold 1 and plot. We use the Librosa library to load it in with sample rate 32000. Librosa automatically normalises the frequencies.



(a) Full excerpt (b) Zoomed in

Figure 1.3: Frequencies vs. Frame number

We clearly see the 5 peaks corresponding to the 5 gunshots heard in the excerpt. When we zoom in we see that the entire thing is simply a sequence of frequencies.

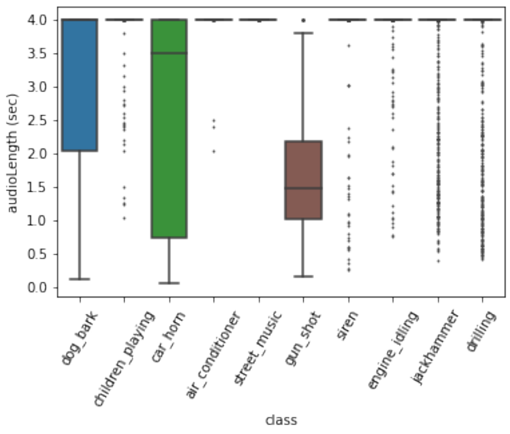


Figure 1.4: Audio Length by class

We can also analyse the audio length. We know from the paper[72] that the maximum length is 4 seconds. We take the length of a sample by subtracting the start time from the end time. We group on classes and graph a box and whiskers plot for each. Our result is shown below.

It is clear audio length is non-uniform. We see the majority of classes have their median at 4 seconds with outliers underneath. Car horn and gunshot however have much lower medians. Car horn shows the widest range of lengths, with dog_bark and gun_shot showing the second and third widest. This could lead to difficulty in predicting these classes, as length will not be a feature that can be found. All others have their 95% confidence interval located entirely on the 4 second mark.

Background

2.1 Deep Learning and Neural Networks

In this section we describe the basic framework of deep learning and introduce neural and convolutional neural networks. This background is necessary grounding for understanding Bayesian neural networks. We make reference to the notes from the Data Science module on neural networks and convolutional neural networks[87].

Deep learning is a type of machine learning based on artificial neural networks. Neural networks are essentially an assembly of neurons organised into layers. Neurons, and neural networks in general, were originally loosely based on the learning of cells in the human brain. These are the building blocks of neural networks. Neurons in the context of machine learning are mathematical nodes, that take inputs from the previous layer and according to a set of weights and biases output a single value. The inputs, x_j, are multiplied by the weights, w_{ij}, and summed. Then a bias value, b_j, is added, before the whole thing is passed through an activation function, σ. The process of a neuron can be described by the following equation:

$$y_i = \sigma(\sum_j w_{ij}x_j + b_j) \tag{2.1}$$

Each layer in a neural network contains a number of these neurons. A neural network is then a number of these layers stacked together. A simple example is shown in Figure 2.1[91].

Any layer between the output and the input layer is referred to as a hidden layer. The neurons in the hidden layer, h_1 and h_2, take as inputs x_1 and x_2 and output a value according to equation 2.1. The output layer consists of neurons too, but in this example there is only one o_1. Therefore this simple model would output a single value from an input with two features (x_1 and x_2). The inputs for o_1 are the outputs of h_1 and h_2. We could have many more hidden layers here of varying numbers of neurons and the inputs to them would always be the outputs from the neurons in the previous layer. This is always the case. The graph formed

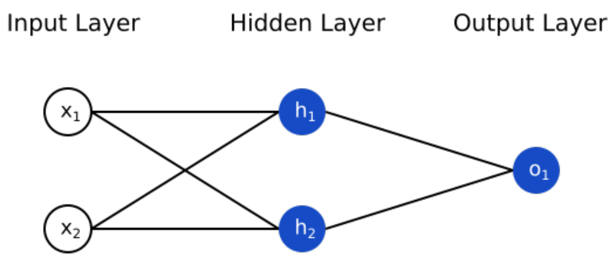


Figure 2.1: Example of a Basic Neural Network

by the network is always acyclic in this sense, with all neurons in a layer being connected to all neurons in the neighbouring layers only. Multi-layer perceptrons (MLPs) are based on the original perceptron, with many hidden layers instead of one.

The first trainable neural network was called the Perceptron and was developed by psychologist Frank Rosenblatt in 1957[70]. We mean trainable in the sense that the weights and threshold used in the step threshold activation function were adjustable. The original Perceptron is somewhat similar to modern neural networks, but only had one hidden layer placed between an output and an input layer. Importantly a learning model was also put forward in the paper along with the Perceptron. This learning model could be shown to be guaranteed to converge for linearly separable data. Despite their similarities, models now are much more sophisticated than the Perceptron, consisting of many layers of varying numbers of neurons with new techniques applied on top.

Training a model means adjusting the weights and biases based on training data in such a way that when unseen (test) data is passed through the model, a good accuracy is achieved. It is done via gradient based optimisation incorporating backpropagation of losses. In order to do this we need a loss function. There are many examples of these. A commonly used one is mean squared error (MSE) loss.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_{true} - y_{prediction})^2 \tag{2.2}$$

Optimisation of the model parameters can be summarised as follows.

1) We make a forward pass, inputting the training data into the model to get outputs.

2) We then calculate the loss based on these outputs.

3) We then calculate the stochastic gradient of the loss function with respect to the model parameters.

4) We update the parameters by subtracting the computed gradient and repeat the process.

We find the stochastic gradient on the loss function by taking a minibatch, some randomly selected subset D_m with $|D_m| = M$, of the whole training data set D_{train}. We calculate the loss on the minibatch:

$$L(\theta, D_m) = \frac{1}{M} \sum_{x_i, y_i \in D_m} l(y_i, f_\theta(x_i)) \tag{2.3}$$

Where θ is the set of model parameters and $l(y_i, f_\theta(x_i)$ is the loss between the prediction and the true value. We then subtract according to $\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t, D_m)$. The hyperparameter η is known as the learning rate and it affects how quickly a model learns. This process is repeated over a predetermined number of epochs.

There are other factors to consider in creating a neural network: The exact error backpropagation method used is quite involved, there are many variants on the stochastic gradient descent optimiser shown here, and there are many regularisation techniques which can be added. Despite this, what we have shown is the basic framework of a deep learning neural network, and provides the necessary background for describing a BNN.

2.2 Convolutional Neural Networks

The neural networks described in the previous section contained fully connected layers. A convolutional neural network (CNN) is similar to a regular neural network, except that it contains layers known as convolutional layers. Similarly to how the original neural network was inspired by the human brain, CNNs were originally inspired by the human visual system. Cells are stimulated by a specific region in the larger visual field[41], which inspired the idea of sequentially focusing on subsets of whole images in order to classify them. Computer vision was and still is a motivation for the developement of these networks. This led to the development of a neural network model named Neocognitron for visual recognition that could "learn without a teacher"[33]. Further research in the field resulted in our modern formulation of a CNN[50], which is trained by backpropogation, similarly to regular neural networks.

The basic premise of convolutional layers is that we pass a convolutional kernel over an input array of some dimensions. The kernel will always be smaller in at least one dimension than the input array it passes over. In this way it fits into the input array, and moves through it horizontally until it reaches the end. Then it moves vertically up a row, and slides through horizontally until the end again. This repeats until the entire input array has been entirely scanned. Linear computations take place at each location. The kernel takes a subset of values (the same shape as the kernel) from the input array and acts on them. The kernel is essentially a matrix of real valued numbers. The action that takes place each time is an element wise multiplication between the kernel matrix and the matrix made up by the subset of the input array. We sum up the resulting multiplied matrix values to find a single value representation of the kernel acting on a specific subset of the input array. Once we have this representation, we move the kernel horizontally onto the next subset of the input array and do the computation again. This move could mean we simply move onto the next horizontal index in the input array and repeat the computation. Alternatively, we can define a stride with which the kernel moves. If this were say set to three, we would move the kernel three indices along the array, column by column, and find single cell representation of the kernel acting on this subset of the input array at each move. This results in an output array of single values, each representing the kernel acting on a different subset of the input array.

This is a general description of how a convolutional kernel in a convolutional layer works. We use the example of a 2-D convolutional layer applied to a 2-D input. This could be a grey scale image, which has one colour channel and thus has two dimensions: height and width. In general, the 2-D convolution on a 2-D input is described by the following formula:

$$(h * k)(i,j) = \sum_{m,n} h(m,n)k(i-m, j-n) \tag{2.4}$$

Where $h(i,j)$ is the $(i-j)th$ element of the matrix defined by subset of the input array. Similarly, $k(i,j)$ is the $(i-j)th$ element of the kernel matrix. This operation is referred to as one convolution. Figure 2.2 shows the action of a 3×4 kernel on a a 7×7 input array[87].

We see the kernel scan over the input array. At each instance, we sum over the matrix found by element wise multiplication between the subset of the input array and the convolutional kernel. We collect these single values to form a 2-D output of shape 5×4. The same idea can be applied for higher dimensions input arrays. For instance, we could have a RBG image, which would have an additional colour channel making our input to the layer 3-D. We could usually use a 3-D kernel in this instance but a 2-D kernel can be used. We could have a 1-D time array, which is what we will be using later in this paper. In this instance, we use a 1-D kernel. Higher dimensional inputs than 3-D are also used. Regardless of the dimensions, the same principles apply.

Usually we apply some padding in a convolutional layer too. This means we "pad" the edges of the input array to ensure the values at the edge of the array are always captured. This is useful when we are using a stride parameter, as described above, of more than one in any dimension. In Figure 2.2, using a stride of (1, 2) would result in every second subset of the input

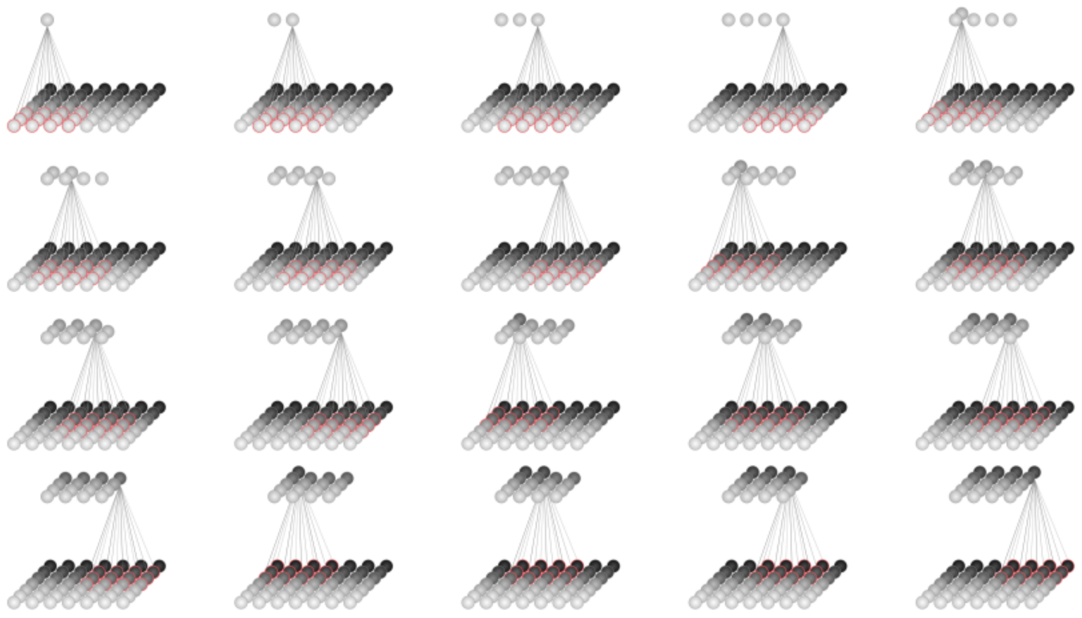


Figure 2.2: Operations inside a Convolutional Layer

array in the diagram being skipped. This results in an output half the size in all dimensions. This would mean the values at the edge (rightmost in the diagram) would never be used in the kernel computations. This is less than desirable as important information on the input could be stored here and we would be disregarding it. This could lead to sub-optimal performance. To get around this, we pad the edges such that these edge values are always included in the feature extraction. There are different types of padding and which type we use must be decided on based on the use case. Reflective padding and zero padding are two common examples.

Figure 2.2 shows the process one convolutional kernel goes through in a CNN layer. Usually, many kernels are used in each layer (32, 64, and 128 being common numbers chosen). Every kernel is optimised with its own matrix values to identify different features in an input. To wrap up our brief introduction to CNNs, we mention activation and batch normalisation layers which are often used in conjunction with convolutional layers. Activation works similarly to that of fully connected layers in regular neural networks, applying a predefined function to all outputs. Batch normalisation normalises over a batch when using stochastic gradient optimisation. It makes for more stable outputs, decreases the effect of overfitting by regularisation, and speeds up training[81]. We will use batch normalisation and ReLU activation layers in our implementation later on.

2.3 Bayesian Neural Networks

In this paper we want to encode a measure of uncertainty into a known neural network. We do this by replacing the deterministic layers with probabilistic ones in order to create a Bayesian neural network (BNN). In this section we explain what a Bayesian neural network is and how to train one. We reference the original paper for Bayes by Backprop[11] and the Coursera course on Tensorflow Probability 2 by Kevin Webster[86] to explain these concepts.

2.3.1 Motivation

Firstly, we provide some motivation behind incorporating a measure of uncertainty in a neural network model. Recently, deep neural networks have permeated almost every walk of life. Use cases are broad and often integral. For instance, deep neural networks are increasingly becoming used in the medical industry with some excellent results coming from research in this area[58][37][68][27]. This is the archetypal example of where uncertainty in neural networks is critical, as a wrong decision or prediction could be extremely costly. Another example is self-driving cars[28][17][3]. Despite these recent successes, there are several drawbacks to neural networks which give pause for thought.

Neural networks generally have a tendency to overfit. This means the model learns the training data too precisely and can mean a model does not predict unseen data well. This issue can be exacerbated if the data the model is trained on is narrow in its variety, very noisy, or contains mistakes. This could mean that the model has not seen large parts of the domain, or that the model will learn these mistakes and noise as if they were meaningful. Also, datasets similar to what is actually encountered in real life can be difficult to compile. Overfitting can be particularly problematic if this is the case.

Another drawback is the lack of transparency in neural networks. Generally a model will tell us its prediction but not how or why it arrived at it. The often used description of a neural network as a black box is fitting in this regard[71]. This lack of deeper insight into how decisions are made has led to mistrust in the algorithms. This has become a controversial topic in America where AI has been used to calculate criminal sentences[61]. Any court has a duty to explain the reasons behind its decisions. When AI plays a big part in a decision however, it becomes difficult to explain.

A third drawback is the susceptibility of neural networks to adversarial attacks make neural networks vulnerable to people who may want to affect them[77]. Researchers have provided a method to create universal, robust, targeted adversarial image patches in the real world[15]. These types of attacks show networks can be easily misled. Unfortunately, these types of attacks are very difficult to defend against. Often, networks confidently make their incorrect predictions in these scenarios. This over confidence is not advantageous and it would in fact be more beneficial for a model to state that it is uncertain.

Providing uncertainty calculations along with predictions enriches the networks output vastly and combats the problems stated above. It could be an integral ingredient in deep neural networks integration into mission-critical tasks in real life.

2.3.2 Types of Uncertainty

Two of the types of uncertainty in neural networks are aleatoric and epistemic uncertainty, which we will describe now. These can both be calculated using a BNN.

Aleatoric uncertainty is data uncertainty. It can be noise, error in measurements or natural randomness or unpredictability of data itself. This uncertainty cannot be reduced and is inherent to the data. It is conceived in the data acquisition stage of creating a model. The paper "A Survey of Uncertainty in Deep Neural Networks"[34] gives an excellent summary of uncertainty and its sources. The factor of error and noise in measurement systems is related to aleatoric uncertainty. This accounts for incorrect or imprecise measurements which clearly lead to uncertainty. For example, in the MNIST dataset there may be an image of a 3 which could be similar to that of an 8. No matter how well the model is assembled and trained this uncertainty will remain.

Epistemic uncertainty is the other category of uncertainty we will calculate. This refers to uncertainty in the model and is reducible, unlike the aleatoric uncertainty. Also unlike the aleatoric uncertainty, epistemic uncertainty is conceived at every stage of the building of a model, from data acquisition through to designing a model and training it. The first factor in relation to epistemic uncertainty is a possible lack of data. This would result in insufficient data for a model to train on. The model would then be uncertain about the values of its weights and biases, coming out in potentially high variances in these weight and bias distributions. This is related to variability in real world situations also. When real world situations change compared to the training set, it can negatively impact test performance and introduce uncertainty into a model. Other factors of the epistemic uncertainty are sub-optimal architecture design, sub-optimal hyper-parameter selection, and errors caused by unknown data[34].

2.3.3 What is a Bayesian Neural Network?

Bayesian neural networks (BNNs) can be used to great effect to model uncertainty in neural networks. As was described in the earlier section, in regular neural networks, we make initialisation estimates of weights and biases. We propagate forwards to make a prediction of based on the training feature data, then use backpropagation in conjunction with this prediction to calculate the stochastic gradient of a predefined loss function with respect to the model parameters. We then update the model parameters accordingly by simply subtracting these gradients. This process is repeated for a given number of epochs. Throughout this process, weights are single point estimates, being updated at every epoch. In some cases, where data is excellent and comprehensive, these estimates may get ever closer to their true optimal value. However optimisation is imperfect in the sense that it does not arrive at an analytical solution so there is always a degree of uncertainty in these estimates. Furthermore, there can be wide ranges of estimates which give good overall results. This point is related to the property of identifiability in a network[65]. In more realistic cases, data contains mistakes, noise and gaps in certain areas. In these cases there is a degree of uncertainty in these estimates that no amount of training epochs will extinguish. Either way, there is a degree of uncertainty in these estimates.

A BNN is a probabilistic variation on the regular neural network that allows us to calculate measures of certain types of uncertainty in predictions. Early work on BNNs dates back as far to 1989[45][82] with further research in the following years[16][54].The basic premise of BNNs is that the point estimates of the weight values are replaced with probability distributions[11]. These distributions have variances, thus encode information on how certain a model is of its weights. Thus, instead of optimising point estimate weight values, we need to optimise the parameters of probability distributions. In the case of a normal distribution on the weights, these parameters would be means and variances.

First, we introduce notation for describing this algorithm and step through some background theory. We represent the set of training examples (or data in general) by D. This consists of a set of features x_i and labels y_i such that $D = (x_i, y_i) = (x_1, y_1), \ldots, (x_n, y_n)$. For regression tasks, $y_i \in \mathbb{R}$, while for classification tasks y_i corresponds to one of a set of classes. We denote the set of model weights by w. We interpret a neural network as a probabilistic model $P(y|x, w)$. Given an input of features x in a classification task, the model maps each possible class of y to a probability using the weights w. As P is a probability distribution, the probabilities of all possible values of y sum to 1. Linear transformations using weights w are responsible for these mappings, the same as described for regular neural networks in the previous section. The weights can be learnt by maximum likelihood estimation, given the data points D. That is:

$$\begin{aligned} w^{MLE} &= \arg\max_w logP(D|w) \\ &= \arg\max_w \sum_i logP(y_i|x_i, w) \end{aligned} \tag{2.5}$$

We define $P(w|D)$ as the posterior distribution of the weights given the training data. It is named as such because it is the distribution on the weights after taking training data into account. We define $P(w)$ as the prior distribution, and this corresponds to our prior belief of what the distribution on the weight should look like. We define $P(D|w)$ as the likelihood of having observed data D given weight w.

The fundamental mathematical idea behind Bayes by Backprop is Bayes' Theorem, which states:

Theorem 2.3.1 (Bayes' Theorem)

$$P(w|D) = \frac{P(D|w)P(w)}{\int P(D|w')P(w')dw'} = \frac{P(D|w)P(w)}{P(D)}.$$

Bayes' Theorem gives us a way to combine a prior belief on the weights with data, or evidence as D defined above is sometimes referred to in the context of this theorem. In the above equation $\int P(D|w')P(w')\mathrm{d}w' = P(D)$. So in order to calculate the distribution of the weights given the evidence, $P(w|D)$, we need to determine this integral over all the possible weights. In practise this is computationally very difficult. We refer to the exact calculation of this as the closed-form analytical solution.

Numerical based methods have been applied as an alternative to this closed form solution. There are many different methods for doing this, but most can be placed into one of two main families: sampling methods and variational inference methods. Sampling methods mostly consist of Monte Carlo Markov Chain (MCMC) methods. MCMC methods are a class of algorithms for sampling from a probability distribution. The samples are created by a continuous random variable with a probability distribution proportional to the function we are approximating. The samples can be used to approximate the integral over the variable then. In this sense we can build a Markov chain with $P(w|D)$ as its stationary distribution, and calculate the complicated integral in the above theorem from this point.

The Metropolis Hastings algorithm is a sampling method[55]. The Metropolis Hastings algorithm is able to sample from the posterior with the knowledge of the likelihood, $p(D|w)$, and the prior probability, $p(w)$ This algorithm considers a Markov Chain of correlated adjacent states which are sampled from the desired posterior distribution (without knowing it's normalizing term). The chain sampled from can be made to be ergodic in such a way that each state is visited a number of times proportional to its probability mass. This method can be used in Bayesian Neural networks[56].

The Hybrid Monte Carlo method is another method of this ilk[60] used for Bayesian learning. It formulates the problem in terms of energy and draws from Hamiltonian dynamics in order to approximate the integral. It splits the problem of sampling from the correct posterior distribution into two sub-problems. In general terms consistent with that in the paper, the aim is to sample as from a probability distribution $P(q)$. In our specific context, q is the weights vector. It is useful to present $P(q)$ in terms of a potential energy function, $E(q)$. This term is chosen such that:

$$P(q) \propto exp(-E(q))$$

The Hamiltonian is defined as the sum of the potential, $E(q)$, and kinetic, $\frac{1}{2}|p|^2$ energies. We can define a joint probability distribution, or phase space, over q and p as follows:

$$P(q,p) \propto exp(-H(q,p))$$

We then take the marginal distribution for q in the above as the distribution from which we want to sample. This is the general outline of the Hybrid Monte Carlo method as well as the Stochastic Dynamics Method from the same paper. Generally, MCMC methods work well as they guarantee producing asymptotically exact samples from the target density[69]. On the contrary, sampling methods are slow and while MCMC methods have been vital in Bayesian statistics in general, when used in BNNs they can be slow for large datasets and complex models.

Variational inference (VI) methods provide an alternative to sampling methods which are much faster[10]. However, these methods do not provide the guarantee of samples being asymptotically exact samples from the target density[69] like the aforementioned sampling methods. Therefore VI is more suited to large data sets and contexts where we want to explore many models quickly, whereas MCMC is suited to smaller data sets and contexts where we are not pressed for time or can afford the extra computational cost to gain more accurate samples. However, data set size is not the only consideration, as the geometry of the posterior to be approximated is also a factor. If the model permits it, Gibbs sampling is a powerful approach to sampling from such target distributions. However this doesn't work on all posteriors. For example, the posterior of a mixture model admits multiple modes and Gibbs sampling isn't always an option. In these situations variational inference may perform better than a more general MCMC technique (like the Hybrid Monte Carlo touched on earlier), even for small datasets [46]. The relative accuracy of VI and MCMC methods is an open question, into which research is still active. It is known that VI methods underestimate the variance of the posterior distribution, but this may be acceptable depending on the circumstances. Some research has shown VI does not suffer in accuracy terms[9][13], while other research has exhibited where it falls short[36]. All in all what is clear is that both families of methods have their advantages and that each are suited to certain context over others.

Because the Urban Sounds 8K dataset is quite large and our model architecture is deep, VI represents the best way to approximate the posterior and build the model. VI methods provide a variational approximation to the Bayesian posterior distribution on the weights. Bayes by Backprop is the method we will use in our implementation later in the thesis, so we describe this here. Essentially, we approximate the actual posterior $P(w|D)$ with another distribution $q(w|\theta)$, where θ is the parameters of this distribution on the weights q. In this way we can decompose the problem of providing the integral in theorem 2.3.1 into an optimisation problem. In order to ascertain how good an approximation $q(w|\theta)$ is to $P(w|D)$, we need some measure of their difference. For this purpose we use the Kullback-Leibler (KL) divergence function:

Definition 2.3.1 *The Kullback-Leibler divergence between two distributions with densities $f(x)$ and $g(x)$ respectively is defined as*

$$KL(f(x)||g(x)) = \int f(x) \log\left(\frac{f(x)}{g(x)}\right) dx.$$

So we want to select the parameters θ such that we minimise the KL divergence between $q(w|\theta)$ and $P(w|D)$. That is:

$$\theta^* = \arg\min_\theta KL[q(w|\theta)||P(w|D)]$$

Going deeper into the KL term in the above equation:

$$\begin{aligned}
KL(q(w|\theta)||P(w|D)) &= \int q(w|\theta)\log\left(\frac{q(w|\theta)}{P(w|D)}\right)\mathrm{d}w \\
&= \int q(w|\theta)\log\left(\frac{q(w|\theta)P(D)}{P(D|w)P(w)}\right)\mathrm{d}w \\
&= \int q(w|\theta)\log P(D)\mathrm{d}w + \int q(w|\theta)\log\left(\frac{q(w|\theta)}{P(D|w)P(w)}\right)\mathrm{d}w \\
&= \int q(w|\theta)\log P(D)\mathrm{d}w + \int q(w|\theta)\log\left(\frac{q(w|\theta)}{P(w)}\right)\mathrm{d}w \\
&\quad - \int q(w|\theta)\log P(D|w)\mathrm{d}w \\
&= \log P(D) + KL(q(w|\theta)||P(w)) - E_{q(w|\theta)}(\log P(D|w))
\end{aligned}$$

But $\log P(D)$ is constant, therefore in order to minimise this KL divergence we need only minimise the latter two terms on the right hand side. That is:

$$\theta^* = \arg\min_{\theta} KL(q(w|\theta)||P(w)) - E_{q(w|\theta)}(\log P(D|w)).$$

And so we are left with the following loss function, which we minimise in our networks:

$$L(\theta|D) = KL(q(w|\theta)||P(w)) - E_{q(w|\theta)}(\log P(D|w)) \tag{2.6}$$

We refer to $KL(q(w|\theta)||P(w))$, which depends on the prior $P(w)$, as the complexity cost. We refer to the second term $E_{q(w|\theta)}(\log P(D|w))$ as the likelihood cost, which depends on the data. The loss function represents a balancing act of satisfying the simplicity of the prior and the complexity of the true data. Intuitively by looking at the loss function, we want the KL divergence between $q(w|\theta)$ and $P(w)$ to be as small as possible, so for them to be as similar as possible. On the other hand, we'd like to maximise the expected log likelihood of $P(D|w)$ with the variational posterior's $q(w|\theta)$ parameters, meaning the model should assign a high log likelihood to the data. As in regular neural networks, finding the parameters which yield the minimum of this loss function is usually computationally impossible. Therefore, we use gradient descent techniques to optimise our model to a satisfactory level. This is another benefit of Bayesian Neural Networks: they are compliant with backpropagation techniques.

Standard optimisation techniques would lead us to find the derivative of $L(\theta|D)$ with respect to θ above, and to update our parameter θ accordingly. Expanding the KL divergence part in equation 2.6, and factoring out the $q(w|\theta)$, we can write the expanded $L(\theta|D)$:

$$L(\theta|D) = \int q(w|\theta)(\log q(w|\theta) - \log P(D|w) - \log P(w))\mathrm{d}w \tag{2.7}$$

We see this could prove quite difficult because the integral over w could be extremely computationally expensive as it would require integrating over the entire parameter w space. To get around this we use a Monte Carlo approximation to calculate the derivatives.

We can rewrite equation 2.7: $L(\theta|D) = E_{q(w|\theta)}(\log q(w|\theta) - \log P(D|w) - \log P(w))$. Essentially what we do is use Monte Carlo sampling to evaluate the expectations in this equation. We then use unbiased estimates of the gradients of the loss function to optimise our parameters in the network, thus learning the variational posterior distribution over the weights. This process is the Bayes by Backprop algorithm. The reparameterization technique required for this moves the dependence on θ so that an expectation may be taken independently of it. It can be shown by the following example in Equation 2.8. Here, $q(w|\theta)$ is a Gaussian, so that $\theta = (\mu, \sigma)$, and for arbitrary $f(w; \mu, \sigma)$:

$$\begin{aligned} E_{q(w|\mu,\sigma)}(f(w;\mu,\sigma)) &= \int q(w|\mu,\sigma) f(w;\mu,\sigma) \mathrm{d}w \\ &= \int \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(w-\mu)^2\right) f(w;\mu,\sigma) \mathrm{d}w \\ &= \int \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\epsilon^2\right) f\left(\mu+\sigma\epsilon;\mu,\sigma\right) \mathrm{d}\epsilon \\ &= E_{\epsilon\sim N(0,1)}(f\left(\mu+\sigma\epsilon;\mu,\sigma\right)) \end{aligned} \tag{2.8}$$

Here, we used the change of variable $w = \mu + \sigma\epsilon$. Now the dependence of $\theta = (\mu, \sigma)$ is inside the arguments of f so we can take the derivatives directly as seen in Equation 2.9 and equation 2.10:

$$\frac{\partial}{\partial\mu} E_{q(w|\mu,\sigma)}(f(w;\mu,\sigma)) = \frac{\partial}{\partial\mu} E_{\epsilon\sim N(0,1)}(f\left(w;\mu,\sigma\right)) = E_{\epsilon\sim N(0,1)} \frac{\partial}{\partial\mu} f\left(\mu+\sigma\epsilon;\mu,\sigma\right) \tag{2.9}$$

$$\frac{\partial}{\partial\sigma} E_{q(w|\mu,\sigma)}(f(w;\mu,\sigma)) = \frac{\partial}{\partial\sigma} E_{\epsilon\sim N(0,1)}(f\left(w;\mu,\sigma\right)) = E_{\epsilon\sim N(0,1)} \frac{\partial}{\partial\sigma} f\left(\mu+\sigma\epsilon;\mu,\sigma\right) \tag{2.10}$$

The final piece to this is the Monte Carlo estimate. In the following Equation 2.11, as i tends to infinity the right hand side tends to the true expectation. This provides us with a good approximation. We can write:

$$E_{\epsilon\sim N(0,1)} \frac{\partial}{\partial\theta} f\left(\mu+\sigma\epsilon;\mu,\sigma\right) \approx \sum_i \frac{\partial}{\partial\theta} f\left(\mu+\sigma\epsilon_i;\mu,\sigma\right), \qquad \epsilon_i \sim N(0,1). \tag{2.11}$$

These equations for the reparameterization technique and the Monte Carlo estimate give us all the necessary tools for calculating the gradients needed for Bayes by Backprop. Previously we had written $L(\theta|D) = E_{q(w|\theta)}(\log q(w|\theta) - \log P(D|w) - \log P(w))$. Under this Monte Carlo approximation, we can rewrite this:

$$L(\theta|D) \approx \sum_i (\log q(w^i|\theta) - \log P(D|w^i) - \log P(w^i)). \tag{2.12}$$

Where w^i denotes the i^{th} Monte Carlo sample taken from the variational posterior $q(w^i|\theta)$. Thus, the loss depends on the samples drawn and which have some variance as opposed to using the analytical KL divergence in the function. Putting together all of these components, we find:

$$f(w; \mu, \sigma) = \log q(w|\mu, \sigma) - \log P(D|w) - \log P(w) \tag{2.13}$$

$$\frac{\partial}{\partial \mu} L(\theta|D) \approx \frac{\partial}{\partial \mu} L(\mu, \sigma|D) \approx \sum_i \left(\frac{\partial f(w_i; \mu, \sigma)}{\partial w_i} + \frac{\partial f(w_i; \mu, \sigma)}{\partial \mu} \right) \tag{2.14}$$

$$\frac{\partial}{\partial \sigma} L(\theta|D) \approx \frac{\partial}{\partial \sigma} L(\mu, \sigma|D) \approx \sum_i \left(\frac{\partial f(w_i; \mu, \sigma)}{\partial w_i} \epsilon_i + \frac{\partial f(w_i; \mu, \sigma)}{\partial \sigma} \right) \tag{2.15}$$

$$f(w; \mu, \sigma) = \log q(w|\mu, \sigma) - \log P(D|w) - \log P(w) \tag{2.16}$$

where $w_i = \mu + \sigma\epsilon_i$, $\epsilon_i \sim N(0, 1)$. With this we arrive at a step-by-step optimisation process for minimising the loss function:

1. Sample $\epsilon_i \sim N(0, 1)$.
2. Let $w_i = \mu + \sigma\epsilon_i$
3. Calculate

$$\nabla_\mu f = \frac{\partial f(w_i; \mu, \sigma)}{\partial w_i} + \frac{\partial f(w_i; \mu, \sigma)}{\partial \mu} \qquad \nabla_\sigma f = \frac{\partial f(w_i; \mu, \sigma)}{\partial w_i} \epsilon_i + \frac{\partial f(w_i; \mu, \sigma)}{\partial \sigma} \tag{2.17}$$

4. Update the parameters with some gradient-based optimiser using the above gradients.

This is how we learn the parameters of the distribution for each neural network weight. The above steps are precisely what is known as Bayes by Backprop.

Finally, it is standard practice to use minibatches in training neural networks, Bayesian is no different. Our loss function can be written:

$$L(\theta|D) = KL(q(w|\theta)||P(w)) - \sum_{j=1}^{N} \log P(y_j, x_j|w_j)$$

where N is the number of training points in the data and $w_j = \mu + \sigma\epsilon_j$ is sampled using $\epsilon_j \sim N(0,1)$. We assume one sample is taken for simplicity. However, when we use batches of size B this becomes: $L(\theta|D) = KL(q(w|\theta)||P(w)) - \sum_{j=1}^{B} \log P(y_j, x_j|w_j)$. B is usually much smaller than N so clearly there is a discrepancy in the ratios of the two terms in this equation. One has reduced the number of terms it sums through while the other hasn't, giving the unchanged term more weight in the calculation. To resolve this, we multiply in a correction factor $\frac{N}{B}$ to the second term to ensure that its expectation is the same as before. This leads to the loss function, after dividing by N to take the average per training value, of

$$L(\theta|D)_{batch} = \frac{1}{N} KL(q(w|\theta)||P(w)) - \frac{1}{B} \sum_{j=1}^{B} \log P(y_j, x_j|w_j). \tag{2.18}$$

Which concludes all our necessary theory on Bayesian Neural Networks. We have comprised all the required elements to build these networks, and explained the theory which underlies them. Now we are ready to implement these theories.

M18 Model

As was mentioned in the introduction, the idea of this thesis was to select a deterministic model from the existing literature and to apply these Bayesian methods to it. As we seen from the research up to now examined in the earlier section “Prior Work”, there are a wide array of models to choose from for this task. Many of these models are quite complex, incorporating transfer learning, heavy data augmentation, ensemble methods and data enhancement techniques. As none of these things are the focus of this paper however, we elected to focus on finding a recorded model that uses limited preprocessing, is both completely trained and tested on a dataset readily available to us, computationally feasible for our resources, and attains close to state of the art results. With this in mind, we will focus on the model found in the work by Wei Dai, Chia Dai, Shuhui Qu, Juncheng Li, and Samarjit Das, 2016[22]. This model takes as its input raw wave forms with some preprocessing and no data augmentation. For computational speed, the audio waveforms were down-sampled to 8kHz and standardized to have mean 0 and variance 1. The paper itself analyzes the effect of deepness of neural networks on performance in the UrbanSounds8K dataset. We will use the same datset and architecture used in the paper, changing only the hyperparameters. The best performing model in the paper was the M18 model, which has 18 convolutional layers. Holding out the official fold 10 to be our test set, with the rest being used for training and validation, a test accuracy of 71.68% was reported. We reproduced this result for the purpose of this thesis. The architectures couple very small receptive fields in middle and later convolutional layers with larger receptive fields in the first layer, chosen based on the audio sampling rate to mimic bandpass filter. It was found in the paper that the 18-layer model (M18) matches the performance of similar models using log-mel features. The full architecture of M18 is as follows. Conv Layer: $[x, y]$ represents a 1-D convolutional layer with kernel size x and y filters. All convolutional layers are followed by batch normalization layers, which are omitted to avoid clutter.

M18
Input: 32000x1 time-domain waveform
Conv Layer: [80, 64] (stride = 4)
Maxpool: 4x1 (output: 2000 $\times$ 64)
$\left.\begin{array}{l} Conv\ Layer : [3,64](stride = 1) \\ Batch\ Normalisation \end{array}\right\}$ x 4
Maxpool: 4x1 (output: 500 $\times$ 64)
$\left.\begin{array}{l} Conv\ Layer : [3,128](stride = 1) \\ Batch\ Normalisation \end{array}\right\}$ x4
Maxpool: 4x1 (output: 125 $\times$ 128)
$\left.\begin{array}{l} Conv\ Layer : [3,256](stride = 1) \\ Batch\ Normalisation \end{array}\right\}$ x4
Maxpool: 4x1 (output: 32 $\times$ 256)
$\left.\begin{array}{l} Conv\ Layer : [3,512](stride = 1) \\ Batch\ Normalisation \end{array}\right\}$ x4
Global average pooling (output: 1 $\times$ 512)
Softmax: 10 classes

3.1 Important Factors in our Probabilistic model

Using the above architecture as out starting point, we sought to equal the performance of the deterministic M18 with a probabilistic M18. We will refer to the Bayesian Neural Network we constructed based on M18 as M18-P (where the P signifies "probability"). In this section we analyse certain hyperparameters in optimising M18-P. We will describe these parameters and provide statistical evidence on their impact on the performance of M18-P. All code written to generate experiment results and to generate graphs was written by the author, and can be found here[25]. All results are reproducible.

There a number of ways to optimise a hyperparameter of a neural network. The goal is to find a set of hyperparameters which give the optimal performance in relation to the loss function. We generally hold out a validation set on which we test these hyperparameters. This is a subset of the training set that is separate to the testing set. These hyperparameters can be interdependant[74], so finding the optimal combination can be computationally expensive. There are several ways to approach the problem[6]. The most obvious way to do this would be to simply search through every possible combination of hyperparameter values in a subset of the parameter space. This parameter space would be defined before hand. This method is referred to as grid search. Since many hyperparameters, like the learning rate for example, are real valued, a reasonable range and intervals to test on within this range must be defined before carrying out the grid search. The search suffers from the curse of dimensionality, which refers to problems that occur in higher dimensions than everyday 3 dimensional experience. Every number of hyperparameters we want to test is multiplied together, so clearly a higher number of hyperparameters can lead to this method becoming computationally expensive very quickly.

An alternative to this is random search. It uses a grid with a dimension and a range for each hyperparameter, similarly to the grid search. Instead of exhaustively testing every combination of hyperparameter values, we randomly select hyperparameter values in the ranges and test on these. This cuts the computation time considerably. This method has been shown to perform as well if not better than grid search in a fraction of the runtime[7]. It allows the inclusion of prior knowledge if we have any of what the hyperparameters might look like, as we can specify the distribution of each hyperparameter from which to sample.

Other methods for hyperparameter selection include Bayesian optimization[79], Gradient-based optimization[5], and Evolutionary optimization[12] to name but a few. The method we will use is a variation on the grid search. We find a reasonably performing model through manual search first. Then we create a grid of values, values in a chosen range of real numbers or a selection of categorical variables. We search through the grid on one dimension (for one hyperparameter) keeping all other hyperparameters constant. We use this process to produce the plots below which describe the effect of some parameters. We set the hyperparameter just tested to the optimal value and repeat the same process on the next hyperparameter. We do this until all hyperparameters have been optimized.

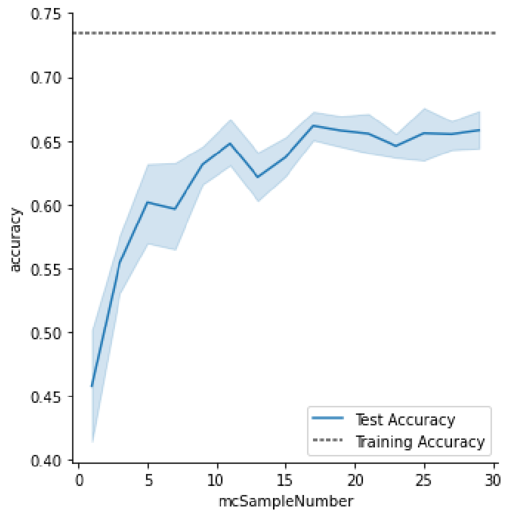
We provide test results on how some parameters related to performance below. The following analysis of hyperparameters was carried out in our implementation of M18-P only.

3.1.1 MC Sample Number

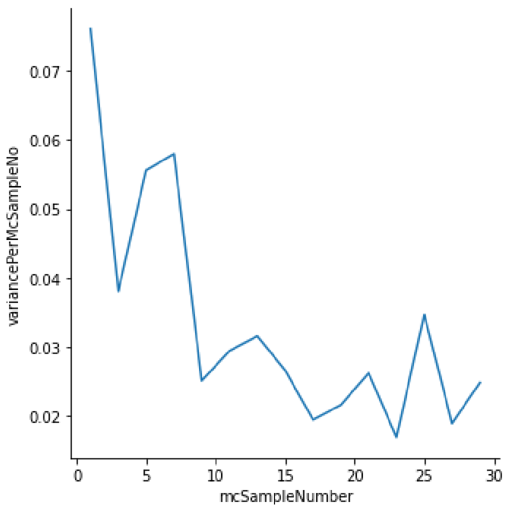
A factor in the test prediction process is the number of MC samples we average over. This is not a factor in training at all, and only concerns making predictions at test time. Each time we pass a data example through a trained model, the distributions on the weights are sampled from. From this point the process is the same as in a regular neural network, where the data undergoes linear transformations that result in a prediction. The combinations of distributions samples are different each time, thus giving different softmax outputs each time. A model trained on insufficient data or extremely noisy data for instance could expect greatly differing soft max outputs as there is likely a high degree of uncertainty in the model's weight distributions. In this sense, a Bayesian Neural Network is in fact an infinite collection of neural networks[11]. We can make more than one forward pass for each single data example, sampling several times from the weight distributions in order to generate several predictions. We can then average over these sub-predictions to get one final prediction, which we evaluate the accuracy against. This number of predictions we average over is referred to as the MC sample number.

In our classification example, we considered two ways to average over these sub-predictions. One could take the modal class from the sub-predictions. That is, simply take the highest probability class from each sub-prediction, and then take the most frequent class over these as our final prediction. Alternatively, one could take the average of the softmax probability distributions of the sub-predictions. This is done by simply adding up the distributions, each vectors of length 10 in our Urban Sounds 8K case, and dividing by the number of predictions per example we calculated. We found the latter method to generally give better performance so opted for this.

To test the effect of the factor MC sample number, we trained a model over 300 epochs. Similarly to the previous section, we set the learning rate to 1×10^{-4} and batch size to 400. We use the scale mixture of two normal distributions for the prior, with $\sigma_1 = e^3$, $\sigma_2 = e^{-7}$, and $\pi = 0.5$. We use a multivariate normal distribution as our variational posterior, and a KL Divergence approximation sample number of 5. Accuracies were calculated on the test set.



(a) Accuracy vs. MC sample number



(b) Variance vs. MC sample Number

Figure 3.1: MC sample number vs variance/test accuracy

Once the model is trained, we begin making predictions. We do not make any reference to the training statistics in these graphs as the MC sample number only affects the test time predictions. For each MC sample number, we evaluate the accuracy over the entire test set 10 times. In this way we can also account for the variance in the accuracies as the MC sample number is increased. It also reduces the effect one outlying evaluation has on the graph. To clarify, for each data point in the test set, we make 10 final predictions. Each of these 10 final predictions are averaged over a number of sub-predictions. This number is the MC sample number, which we vary. We do this to get a truer idea of the effect the MC sample number has on performance, and to also see how variance in accuracies changes with the MC sample number.

Figure 3.1 (a) graphs the accuracy as calculated on the test set versus the MC sample number. We clearly see an improvement in general performance as we increase the MC sample number. The accuracies recorded increase rapidly from MC sample numbers 1 through to ten and then tails off, with only marginal gains thereafter. This is an important result, and gives probabilistic models another parameter with which to get better results. We include in this Figure 3.1 (a) dashed line which represents the training accuracy reached for the process. We show this to show how the gap between test and training accuracy is closed significantly by increasing the MC sample number, but how a gap of less than 10% still remains. Figure 3.1 (b) graphs the variance of each MC sample number. Although the line is not smooth at all, it generally tends to decrease as MC sample number increases. This is an unsurprising result. Generally, the more we sample from any distribution, the closer the average gets to the mean and the smaller the variance over the samples gets. The same is true here, where more samples is equivalent to higher MC sample numbers.

3.1.2 Batch Size

We found that the choice of batch size dramatically affects the performance of the model. Using the validation set, we first found the optimal values for the other hyperparameters. We found a learning rate of 1×10^{-3} gives the best results over 300 epochs. We set the number of samples to take for the KL divergence approximation to be 5, and the number of MC samples to average over for the final prediction to be 10. We chose 10 so as to marginalise the amount of variance in the final predictions, as choosing a smaller number would incur higher variability in the predictions. We chose to use a scale mixture prior as used in the original paper with $\sigma_1 = 1$, $\sigma_2 = e^{-6}$ and $\pi = 0.25$, and a multivariate Normal distribution as the variational posterior. We

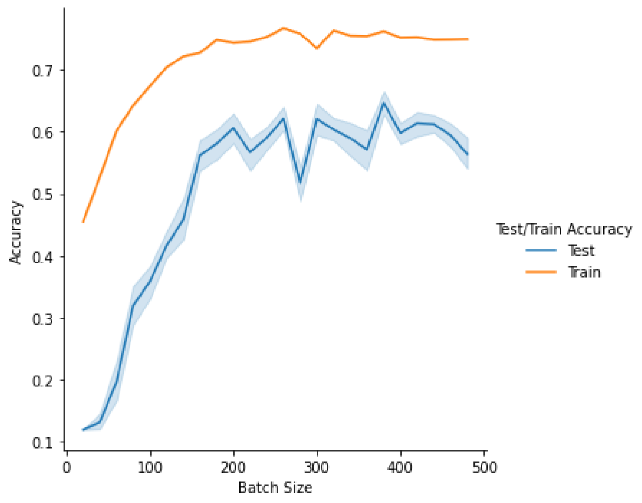


Figure 3.2: Batch Size vs. Test Accuracy

varied the batch size between 20 and 480. For each batch size, we made 10 predictions on the test data. We see the results in Figure 3.2.

In this line graph and all following this, the dark line signifies the mean of all records for a particular value on the x axis. The lighter region around it signifies the 95% confidence interval, calculated based on all the records for a particular value on the x axis. Clearly, there is a strong positive correlation between the two variables. As we increase the batch size, we seen big increases in test accuracy. This rate of increase tails off after batch size of roughly 200. This claim is backed up by the calculated Pearson correlation coefficient, which is a measure of linear correlation between two variables that can take values between -1 and 1. It was calculated to be 0.7608, which is consistent with our claims. We also plot the training accuracy, and we can see the test accuracy follows it pretty closely, although the test accuracy can be a little more erratic.

This is a somewhat unsurprising result. Generally, small batch sizes have a regularising effect (Wilson and Martinez, 2003), while larger batch sizes tend to generalise worse. This can be useful where overfitting is a problem. Here however, it seems to contribute to the model underfitting. This increase in generalisation normally associated with smaller batch sizes is only beneficial if the model is still seeing enough data in each batch to learn effectively. For example, the training accuracy for batch size 20 doesn't break 51% over the 300 epochs. At the other extreme, the training accuracy of batch size 480 consistently exceeds 75%.

The deterministic model ran with similar parameters and a batch size of 32 and achieved test accuracy of 71.68%. There is a huge amount of added stochasticity when adapting to a Bayesian neural network. Without changing the batch size, this added randomness seems to completely throw off the the models learning.

3.1.3 KL Divergence Sample Number

The number of samples taken in the KL divergence approximation function is another factor in our network. As explained earlier, we cannot always compute the analytical KL divergence between two distributions. Furthermore, it has been found that priors with difficult to compute KL divergences performed better[11]. Thus we sample from the variational posterior and the prior in order to calculate this KL divergence approximation. Intuitively, the more samples we average over the better represented the distributions are. We seek an answer as to whether this better representation of the distributions when calculating the KL divergence approximation leads to better performance. Furthermore, how many samples should we take for optimal performance? To gather our data, we ran tests over 300 epochs with a learning rate of 1×10^{-4} and batch size of 400. We use the scale mixture of two normal distributions for the prior, with $\sigma_1 = e^3$, $\sigma_2 = e^{-7}$, and $\pi = 0.5$. We use a multivariate normal distribution as our variational posterior. Accuracies were calculated on the test set. The following figure, Figure 3.3, summarises our results.

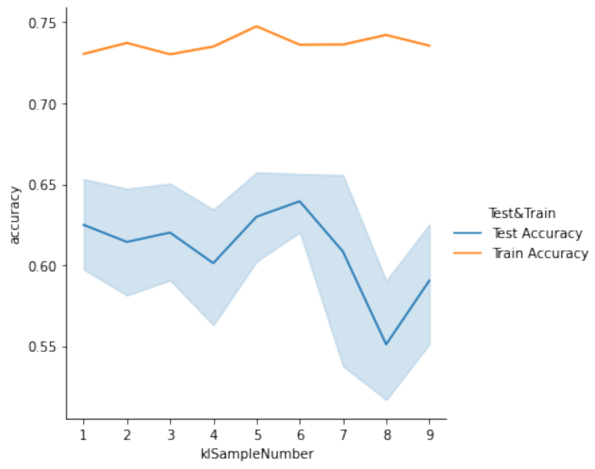


Figure 3.3: KL Divergence Sample Number vs. Test Accuracy

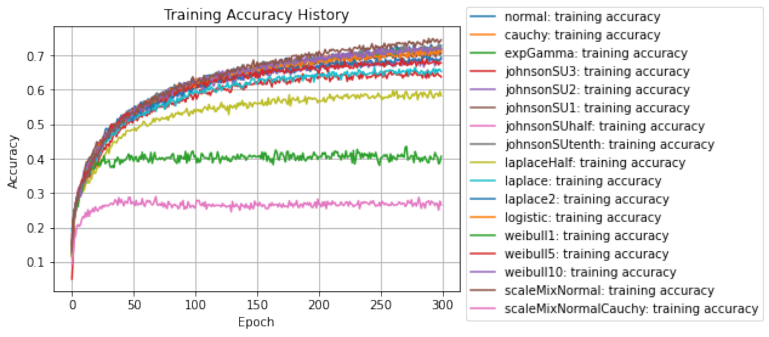
Surprisingly, the number of samples taken in the divergence function does not seem to have any effect on training and test accuracies. Both remain somewhat constant as we increase the number of samples drawn. There is a small drop in test accuracies, but I would attribute this to variance. This suggests that a more precise representation of the prior and posteriors when calculating the approximate KL divergence, obtained by averaging over a higher number of samples, does not improve overall results.

3.1.4 Prior Distributions

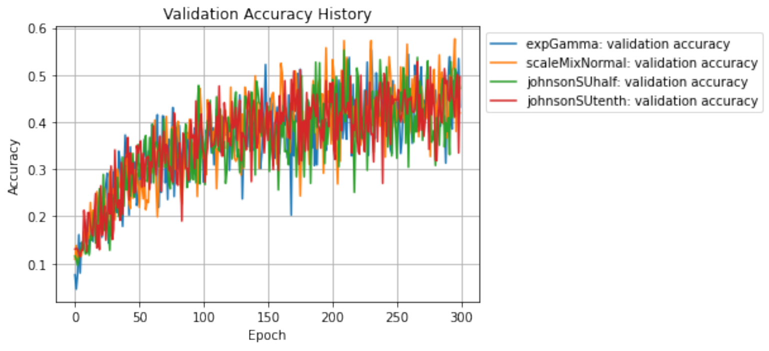
Another important factor in training a Bayesian Neural Network is the choice of prior distribution. Isometric gaussians are the de-facto standard in the field[38][26][52], but research has been done to see whether whether these priors accurately reflect our true beliefs about the weight distributions or give optimal performance[32]. Other research into this field makes efforts in automating the selection as well as explaining why some perform better than others[31]. We present our results on different prior distributions used for training and testing in this section. Following this we provide some interpretation by analysing the distributions themselves.

The prior distribution can be chosen to have learnable parameters also. In this paper however we focus on fixed priors, as per the original paper[11]. The parameters of these distributions do not change throughout the entire process (training or testing). This pushes the learning down to the posterior. As was noted in the original paper, learnable prior distributions often optimise too quickly, sometimes leading to local max parameters and inhibiting the posterior from tending towards a globally optimised set of parameters. The prior we have been using until now is the scale mixture Normal prior. In this section we want to test other priors, to see the effect this has on the results.

In Figure 3.4 we graph the training and validation accuracies for several prior distributions. In our tests we trained over 300 epochs with a learning rate of 1×10^{-4} and batch size of 400. We used the multivariate normal posterior once again as the variational posterior. We use 10 MC samples for testing. Let's first give a quick description on the tested priors. Most of the priors tested have two main parameters: mode and standard deviation. The modes in all priors are set to 0. The distributions with just these parameters are the Normal, Laplace, Logistic, and Cauchy distributions we can see in the legend of (a). We include three different Laplace distributions: laplaceHalf, laplace and laplace2. These refer to the 0 centered Laplace with standard deviations $\frac{1}{2}$, 1, and 2 respectively. The Weibull distributions are arranged similarly to this, except instead of a mode parameter it has a concentration parameter which we keep constant at 1 while we vary the standard deviation. The standard deviations 1, 5, and 10 correspond to weibull1, weibull5, and weibull10 respectively. We also see a list of Johnson SU priors. The Johnson SU distribution is a transformation on the Normal. However, as per the Tensorflow Probability documentation[24], this distribution is defined by four parameters not two: skewness, tailweight, mode, and standard deviation. In all Johnson SU priors we set skewness to 0, mode to 0, and standard deviation to 1. We then vary the tailweight assigning tailweight 3, 2, 1, $\frac{1}{2}$, $\frac{1}{10}$ to johnsonSU3, johnsonSU2, johnsonSU1, johnsonSUhalf, and johnsonSU3tenth respectively. The ExpGamma distribution, not fitting into the descriptions of the earlier distributions, does not have a location or standard deviation parameters. Instead it has rate and concentration, which we set to 1. Finally the scale mixture priors have three additional parameters to the standard deviation and mode of the distributions being mixed. These are σ_1, σ_2, and π. We set these to e^3, e^{-7}, and 0.5 for the scale mixture of the Normal distributions, and e^1, e^{-3}, and 0.75 for the scale mixture of the Normal and Cauchy distributions. More priors were in fact tested but we cannot include all of them.



(a) Training Accuracy



(b) Validation Accuracy

Figure 3.4: Accuracies for different priors

From Figure 3.4, we can see the training and validation histories for the different priors. Only the best performing 4 priors histories are shown for validation accuracy in (b), as otherwise the graph gets very messy. In (a), we notice the difference in the training accuracies they converge to is large, with some under 40% training accuracy and some over 70%. The scale-mixture of two Normal distributions performs marginally the best in training, with a 2% advantage on the next highest. The scale mixture of Cauchy and Normal distributions performs worst however. In our visualisation of the different priors we see the two do not appear very different, as such this appears a strange outcome. We observe the validation accuracy history in (b). We show much less validation plots than we calculated (we calculated validation accuracies for every training accuracy) but because the graph gets very messy after more than two or three plots, we opted to only include the best performing three. There are a number of priors that visibly converge similarly. The different priors and the accuracies they converge to after 300 epochs are summarised in the following table (accuracies were taken as an average of the last 10 recorded:

Prior	Training Accuracy	Validation Accuracy
cauchy	0.7131	0.4394
expGamma	0.7146	0.46856
johnsonSU1	0.70484	0.4353
johnsonSU2	0.6843	0.3728
johnsonSU3	0.6433	0.2930
johnsonSUhalf	0.7190	0.45830
johnsonSUtenth	0.7208	0.4508
laplace2	0.6876	0.3913
laplaceHalf	0.5865	0.2346
laplace	0.6542	0.3224
logistic	0.7090	0.3763
normal	0.7254	0.4340
scaleMixNormalCauchy	0.2678	0.1195
scaleMixNormal	0.7422	0.4633
weibull10	0.7183	0.4408
weibull1	0.4013	0.2197
weibull5	0.6790	0.3839

We now look at the test accuracies amongst the different priors. We graph a box plot with the prior distribution categorical variable on the x axis and the real numbered test accuracy on the y axis in Figure 3.5. To calculate the test accuracies, we took 10 predictions averaging over 10 MC samples each. We see the different ranges for different priors with certain outliers too.

The best priors seem to be the normal, cauchy, johnsonSUtenth, weibull10 (with standard deviation 10) and scaleMixNormal. While the worst are laplace, laplaceHalf, johnsonSU3, weibull1, scaleMixNormalCauchy. Their univariate equivalents are plotted below. The prior represents what we think the distribution on the weights should look like before we start training. This choice of prior somewhat defines which weight parameter values we consider. The parameter space is infinite, and the prior distribution will push the model towards certain

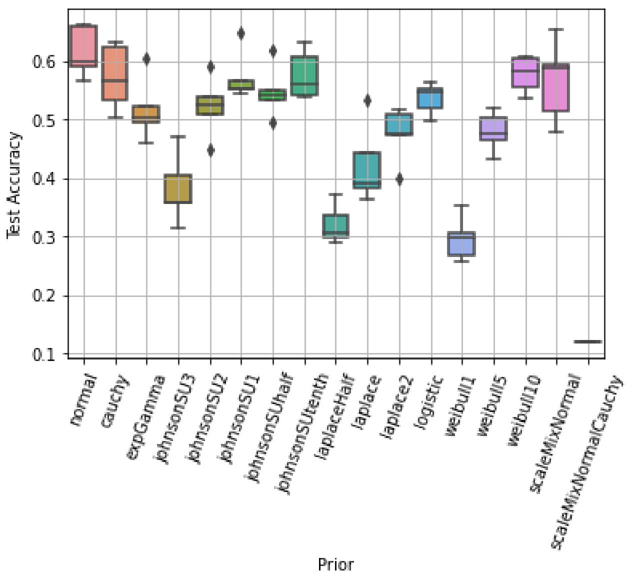
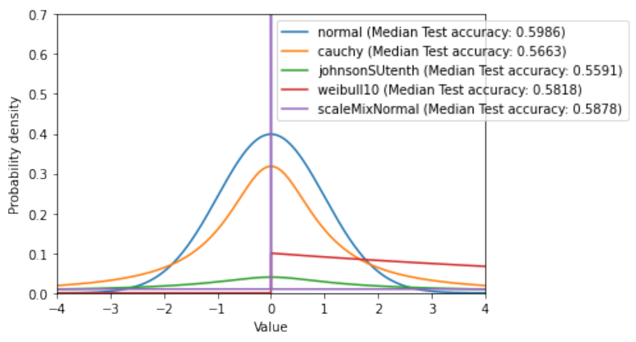


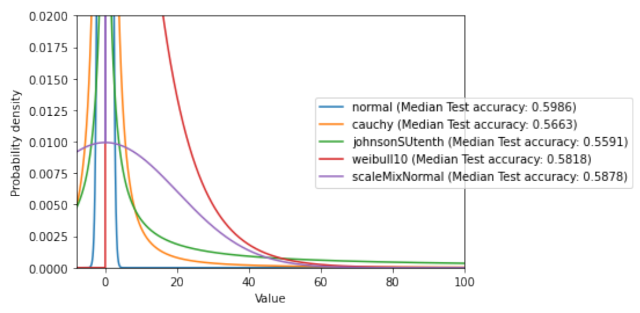
Figure 3.5: Prior Distributions vs. Test Accuracies

values. For instance, in the original paper[11], they use a scale mixture of normal distributions for the prior. This resembles a spike and slab distribution, which has a large "spike" at the mode zero and falls to a "slab" extremely quickly moving away from zero. The spike can have very large probability density, however the slab density is still non-zero thus values not equal to zero are still considered. We want to minimise the KL divergence approximation function, which contains a negative term that calculates the log likelihood that a sample taken from the variational posterior distribution was taken from the prior distribution. Because this term is negative, we want to maximise it in order to minimise the KL divergence. Therefore the model will push the posterior towards taking samples with high log likelihoods in the prior distribution. In the case of the spike and slab prior, this means samples close to zero. This has a somewhat regularising effect. With this intuition in mind, we examine the plots in Figure 3.6 which visualise the best performing priors in our tests.

The two plots, (a) and (b), in Figure 3.6 plot the same set of distributions but we have adapted the axis limits to show certain aspects in each. First of all, it's immediately clear that the priors are very different. All but one are symmetrical, with the Weibull distribution being the exception with zero probability density for all negative numbers. It is a somewhat surprise inclusion as we would have expected the well performing priors to have some density on each side of zero. In the same sense as our intuition on the spike and slab distribution in the previous paragraph, this prior pushes the prior distribution towards positive values. There are also some similarities. All priors peak at 0 and decay at different rates. The scaleMixNormal



(a)

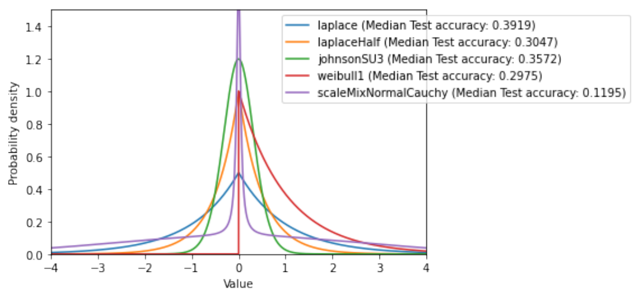


(b)

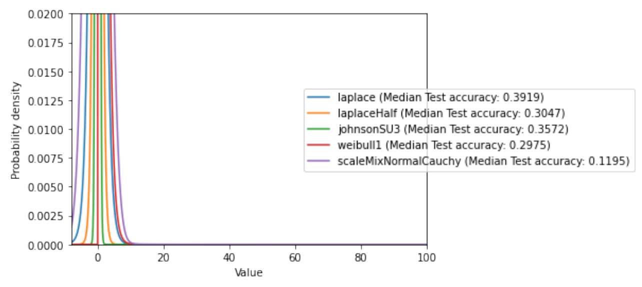
Figure 3.6: Best Performing Priors

prior has a peak of 218.7565, much higher than the others, but we do not zoom out on the graph to show this as that meant we would not see the other distributions. In the (b) we zoom in on the positive y axis and out on the x to show how the distribution tails look. As noted all distributions except for Weibull (which is zero everywhere along $\mathbb{R}^-$) are symmetrical so the tails are identical either side. We see that all distributions maintain a significant density on their tails. This suggests that while zero has the highest density in all, values far beyond this are still considered.

To contrast Figure 3.6 in order to gain some insight into the model parameter, we also plot the worst performing priors.



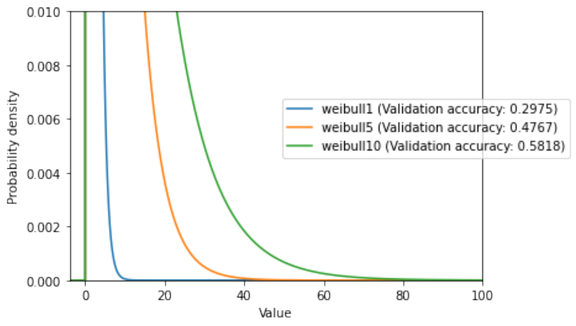
(a)



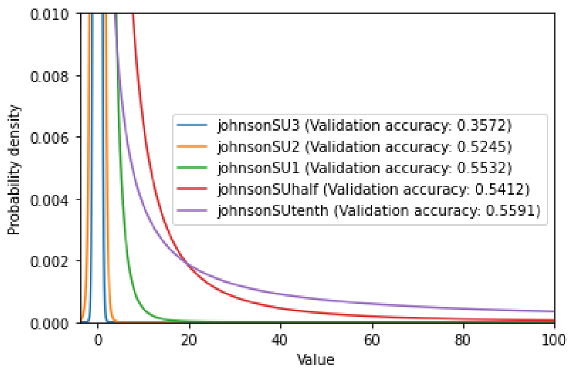
(b)

Figure 3.7: Worst Performing Priors

While all distributions from the best and worst performing priors have peaks at 0, these peaks are different. Despite the peaks in the worse performing priors being generally higher, the scale mixture of the Normal distributions has the highest and is a well performing prior. This implies the peak itself is less important and its the distribution of probability around it we should take as important. We can clearly see by comparing Figure 3.6 and Figure 3.7 that the better performing priors have much more probability on their tails a larger distance away from 0. This implies the model performs better when the prior encourages a wider array of weights as this is what best predicts the dataset in this task. This point is further exemplified by Figure 3.8, where we plot all the Weibull distributions and all the JohnsonSU distributions in (a) and (b) respectively. We can see from the graphs that as the tail weight increases in these distributions, test accuracy increases. This conclusion is consistent with other previous research into priors in BNNs. In the 2021 paper "Bayesian Neural Network Priors Revisited"[32], it was found that heavier tailed prior distributions generally led to better results as taken by summary statistics. A Bayesian CNN was trained and tested on the) on MNIST dataset, like what we have in our experiments.



(a) Weibull Distributions



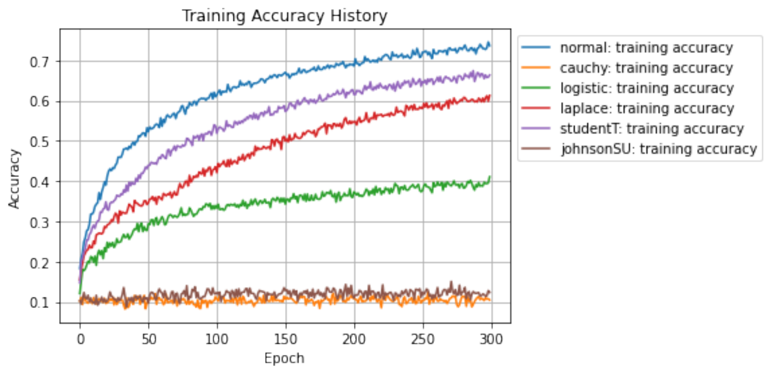
(b) JohnsonSU Distributions

Figure 3.8: Weibull and JohnsonSU Dsitributions

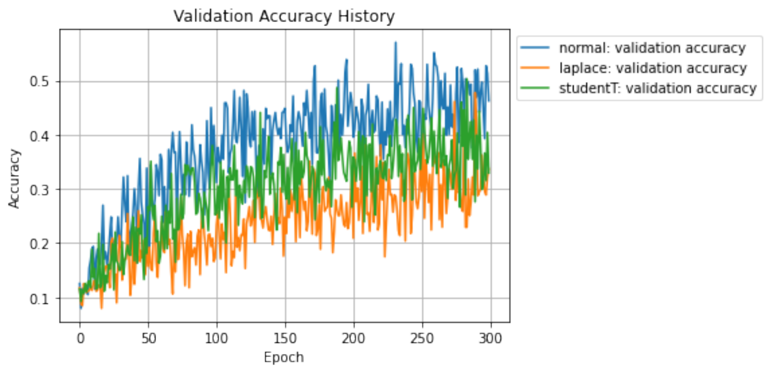
As an aside, accuracy and prior choice are two variables discussed here, but the choice of posterior is an important factor too. We may get very different results for the same priors had we chosen a different posterior. The loss function depends on how these two distributions interact. The prior can push the posterior towards certain parameters, but the posterior is of fixed form so this can occur very differently depending on this form. However, it would be too time exhaustive to test every prior with every posterior and to provide results. The purpose of this was to show the different priors and how the choice impacts on the model results. We have shown that prior distributions with larger tail weights better describe the weight distributions, leading to better performance evaluations. This result is consistent with other research into Bayesian CNNs for classification.

3.1.5 Variational Posterior Distributions

The final factor we will consider is the posterior in our model. This distribution has trainable parameters that are updated as the training goes on. We will analyse it similarly to the prior choice, by keeping all other hyperparameters constant and tracking the performance of different choices. The final note we made on the prior selection was that the results depended on the couple posterior and prior. So the same points made about their interdependence apply here too. We used the same hyperparameters as in the previous section along with scale mixture of two Normal distributions as prior, and varied the posterior distribution. The forms of variational posteriors distributions we use are Normal, Cauchy, Logistic, Laplace, studentT, and JohnsonSU. We keep the skewness and tailweight parameters set at 0 and 1 in the JohnsonSU distribution, and train the mode and standard deviation parameters. Similarly, we keep the degrees of freedom set at 6 in the Student T distribution and train on mode and standard deviation.



(a) Training Accuracy



(b) Validation Accuracy

Figure 3.9: Training and Validation History

As we can see in Figure 3.9, the choice of posterior distribution can also have a huge effect on the model training performance. Unlike all the prior choices, which gave training accuracies of over 25%, some of the posterior choices result in the model not learning the data at all; that is, the training accuracy remains at around 10% consistently. This suggests the model is selecting at random, or selecting the same label for all examples. The posterior choices that do not train are the multivariate JohnsonSU and Cauchy distributions.

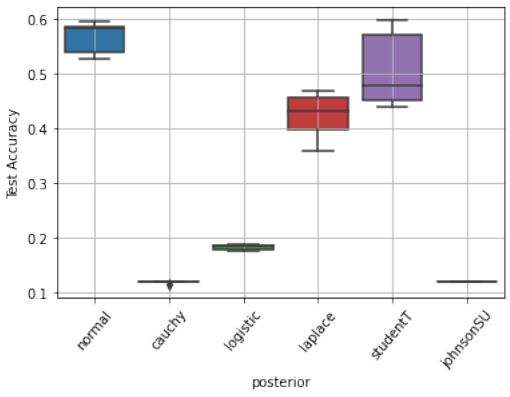
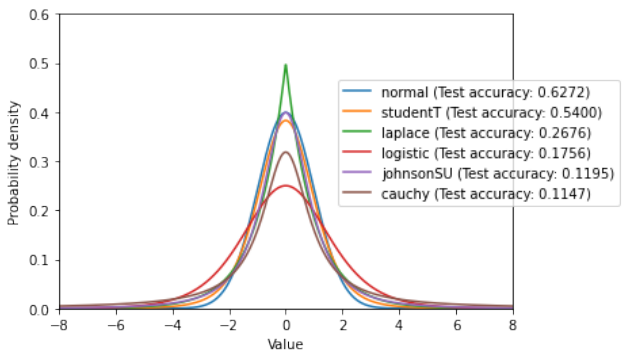


Figure 3.10: Test Accuracies

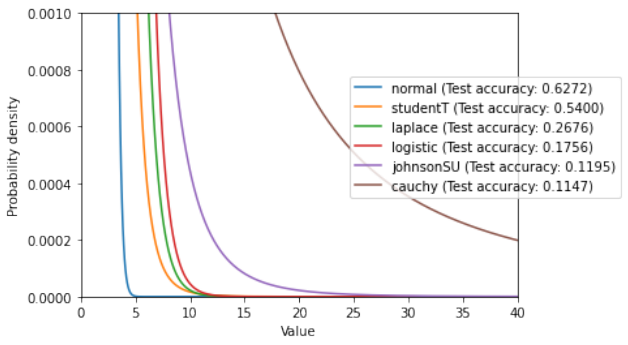
Similarly to in the last section, we create box-plots for the test accuracies when using each distribution. We once again use 10 MC samples to make predictions. We generate the box-plots by considering 10 predictions on the entire test set for each. Our results are shown in Figure 3.10. Unsurprisingly based on the training histories, the Cauchy and JohnsonSU posteriors provide the worst test results, while the Logistic distribution also performs poorly. The ranges of the other posteriors are noted too. The Normal distribution choice performs the best.

As in the previous section, we can plot these distributions in the univariate forms. We set all modes and standard deviations to 0 and 1. This will give us an idea of the shape the distributions, despite knowing that their parameters will be optimised in the learning process.

It is clear to see from Figure 3.11 that the less tail weight a distribution has the better it seems to perform. The zoomed in image of the distributions in (b) in particular implies this. This suggests despite having the ability to learn a standard deviation parameter some of these distributions inherently have too much probability weight on their tails to produce comparative results to the best performing posterior: the Normal variational posterior. This could be related



(a) Zoomed out



(b) Zoomed in

Figure 3.11: Posterior Distributions

to the point made in section 3.1.4 about minimising the KL divergence between the posterior and prior, as a distribution with a greater tailweight may be difficult to optimise towards a slab and spike distribution which has its large "spike" in probability at zero. This concludes our discussion on the importance of MC sample number, batch size, KL divergence sample number, prior distribution, and variational posterior distribution as parameters in a BNN.

3.2 Performance Comparison: Deterministic and Probabilistic

Using the knowledge gained in the last section, we build a BNN called M18-P that produces comparable results to the deterministic model M18. The deterministic model M18 in the original paper was trained on machines equipped with a Titan X GPU, which we don't have access to. We replicated the deterministic model on our own machines in order to get a fair comparison. All models were trained and tested on TU102 [GeForce RTX 2080 Ti]. We summarise the hyperparameters used in the following table.

	M18	**M18-P**
Epochs	300	500
Batch Size	32	400
Learning Rate	1×10^{-4}	1×10^{-4}
L2 Weight Regularisation	0.0001	N\A
MC sample Number	N\A	50
KL Divergence Samples	N\A	5
Prior	N\A	Scale Mixture of two Normal Distributions
Variational Posterior	N\A	Normal Distribution

The two Normal distributions in the scale mixture prior have standard deviations e^3 and e^{-7} respectively, and with mixture ratio $\pi = 0.25$. All hyperparameters were found by first using manual search to find a reasonable model, then sequentially fine tuning each over a reasonable range while keeping all hyperparameters constant, as described in the introduction to the previous section. The breakdown of the parameter in each model is given below.

	M18	**M18-P**
Trainable Parameters	3,683,786	7,359,764
Non-trainable Parameters	7,808	7,808
Total Parameters	3,691,594	7,367,572

So roughly twice the parameters as we expected. The deterministic model ran at an average time of 8s 136ms per epoch. Somewhat surprisingly because of the near doubling of parameters, it was only marginally faster than the probabilistic M18-P model, which ran at an average time of 8s 439ms per epoch. This is surprising at first, but not when we look a little closer. The deterministic model M18 requires quite a small batch size of 32 in order to generalise well to the test data. The training accuracy never drops below 99% after 50 epochs, and it requires a small batch size to provide some regularisation in order to avoid overfitting. By comparison, in the probabilistic method we require a large batch size to even out the added stochasticity from replacing point estimates with distributions. This greatly reduces the runtime per epoch. However, we found we were able to replicate the deterministic results from the paper in 300 epochs, where highest test accuracy achieved was 71.68%. In the probabilistic model however, we required 500 epochs. All in all this led to the probabilistic model needing an extra 965 seconds to train. Additionally, at test time we run for 50 MC samples, which is essentially 50 forward passes, therefore 50 times the test time of the deterministic model (the calculation of the average of these 50 MC samples is negligible). The computational times are summarised in the following table.

	M18	**M18-P**
Time/Epoch	8s 136ms	8s 439m
Total train Time	2448s	3413.8

The deterministic model M18 gives between 70% and 71.6% test accuracy consistently when we tested over several train times. Our best M18-P probabilistic model however gives different accuracies test accuracies within one train time. We found the model with hyperparameters described at the start of this section achieves an average test accuracy of 67.28%. We found this by training the model, and making 20 predictions. Of these accuracies, the minimum was 64.3%, and the maximum at 70.074%.

	M18	**M18-P**
Test Accuracy	71.68%	67.28

While the accuracy of M18-P is comparable to that of M18, there is a definite drop off in performance. Despite testing many combinations of prior and posterior distributions and a wide array of hyperparameters through the methods described earlier, it's possible our model hyperparameters are sub-optimal. It was impossible to test more combinations and ranges of hyperparameters for the purpose of this thesis however due to time restrictions, but perhaps further research using the general trends found on parameters in Section 3.1 could lead to a better performing BNN. What we have shown concretely is that we can achieve comparable results using probabilistic methods.

3.3 AUROC Curves

An important measure of performance for a classifier is the AUC ROC curve. ROC stands for Receiver Operating Characteristics, while AUC stands for Area Under The Curve. It is also referred to as AUROC. It is usually used for binary classifiers but can be extended to multi-class situations by simply choosing one class to be positive and taking the rest to be negative. In this way, for a ten class dataset we could generate ten curves. It helps us visualise the performance of a model by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. TPR is also referred to as recall or sensitivity, and can be thought of as probability of detection. FPR can be thought of as probability of false alarm, and is calculated as 1-sensitivity, where sensitivity is True Negative Rate (TNR). Sensitivity refers to the proportion of examples labelled negative when they were in fact negative. The AUC score is a value between 0 and 1. An AUC score close to 1 indicates good separability in the model predictions. That is the model is at predicting negative examples as negative, positive examples as positive. An AUC score close to zero indicates the model is poor in this respect[40].

Taking each of the classes in turn as the positive class and making the rest negative, we plot the ROC curve and calculate the AUC score for each. These are plotted in Figure 3.12 along with AUC score. For every class, the AUC score is quite high, with the minimum coming in at roughly 0.731568. This score was recorded taking for siren class, suggesting the model

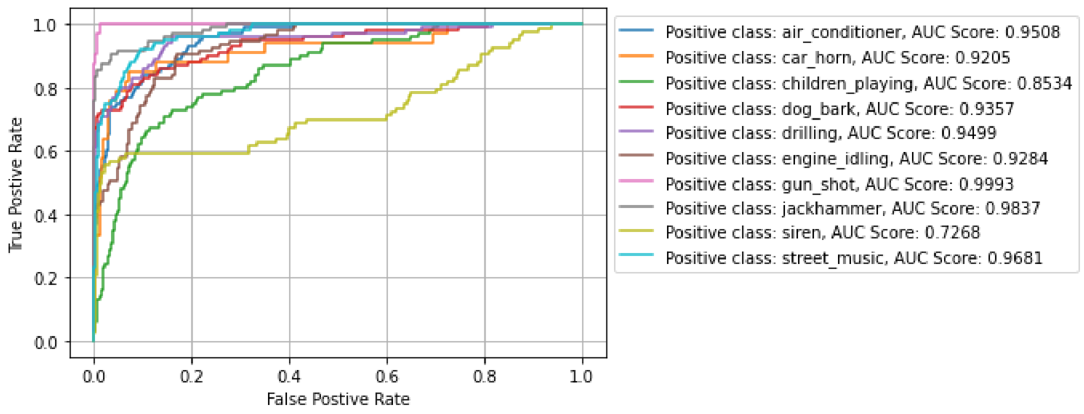


Figure 3.12: AUROC Curves for each class

predicts significantly more false negatives and false positives for this class than other classes. The next lowest AUC score was 0.8534 is for the children_playing class. Clearly the model has much more difficulty with these two classes than any other. On the contrary the gun_shot class scores a very high AUC score. This implies that if the model predicts an example to be in class 6, it is almost certainly the case according to this test set. All in all the AUROC curves are generally positive, and we can say generally that if the probabilistic M18 were to predict a particular class, we could be reasonably confident of this being correct. This is particularly true for some classes, and less so for others.

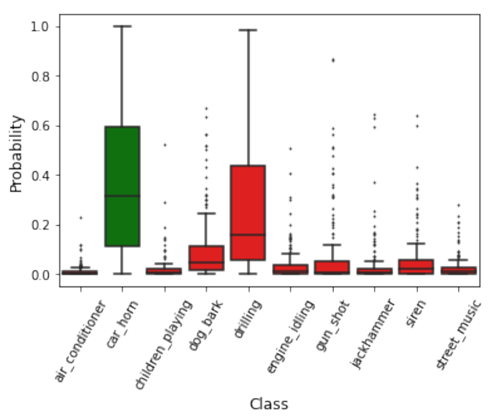
3.4 Plotting Uncertainty

Despite the probabilistic model generally giving slightly less accuracy, the trade off is that we can calculate uncertainty on predictions. We can visualise the epistemic uncertainty by making more than one soft max prediction for a particular test example and graphing a box plot. Because the model samples from weight distributions at each forward pass, the same example will generate a different categorical distribution output each time. The box plot summarise these results, indicating to us the median probability, interquartile range (IQR), and outlier data on each category class. Clearly, the less certain the mode weights are, the more variability we will see in the outputs of the same inputs. We produce the visualisations for 4 selected test examples. For each test example below, we made 200 predictions of the 10 class softmax distribution. We see the 10 possible classes along the x axis, along with the probabilities calculated for each of them on the y axis. Each class corresponds to a range of probabilities. The true category class is indicated by the green box. We chose these four test examples for their varying nature. We will also generate bar charts for specific test examples which indicate the distribution of classes predicted over the 200 instances. We will carry out these visualisations using both 1 and 20 MC samples, in order to observe the difference. We chose the examples from the test set specifically to show certain behaviours.

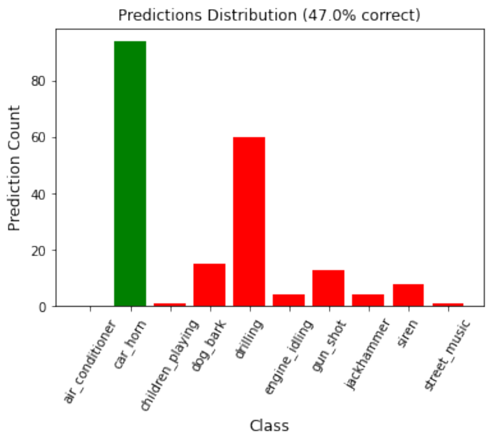
3.4.1 Test Example 1

MC Samples: 1

In Figure 3.13 (a) we see the IQR of the correct class, car_horn, is large. While its maximum is almost perect close to 1 the IQR is located between 0.1 and 0.62. The classes dog_bark, siren, and drilling also have significant ranges despite being incorrect. We also see a huge number of



(a) Predicted Probabilities



(b) Predicted Classes

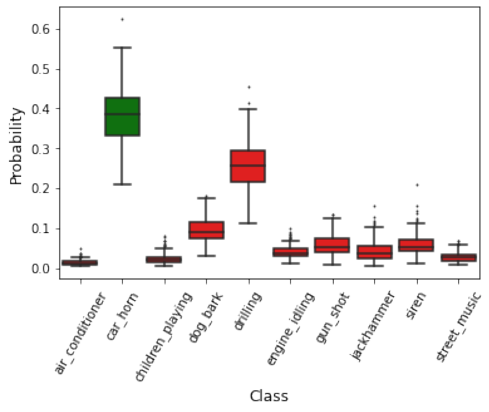
Figure 3.13: Uncertainty Plots: Example 1: 1 MC Sample

outliers, where even classes with a low IQR can be predicted with probabilities as high as 0.9. this is evident with the jackhammer class, where high outliers show taking 1 MC sample with this model can occasionally result in over confident, incorrect predictions. This suggests a high level of uncertainty in these probability predictions. This translates to the predicted classes in (b), where we see the correct class is only predicted correctly 50% of the time. Unsurprisingly from (a), dog_bark is predicted about 25% of the time while the rest of the predictions are spread out amongst all other classes. Every class is predicted at least once, suggesting the model is very uncertain on this test example.

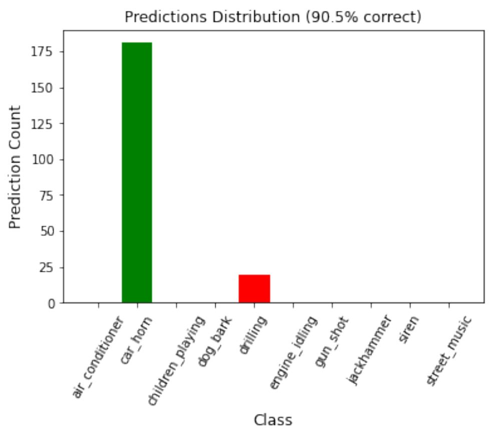
MC Samples: 20

On the same test example, we make 200 predictions again but this time use 20 MC samples each. The corresponding plots are seen in Figure 3.14.

The contrast is stark. Using 1 MC samples the uncertainty in the model predictions was clear to see, here it greatly reduced. The IQRs of the two most probably predictions in Figure 3.14 (a), car_horn and dog_bark, have been reduced. The large number of outliers is also visibly taken care of. The maximum probability apportioned to all classes, including the correct one, is brought closer to the IQR. This results in the correct class being predicted 90% of the time for 20 MC samples, as opposed to 50% for 1 MC sample. This is a significant improvement.

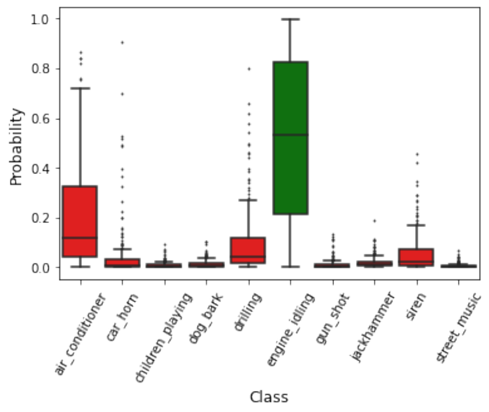


(a) Predicted Probabilities

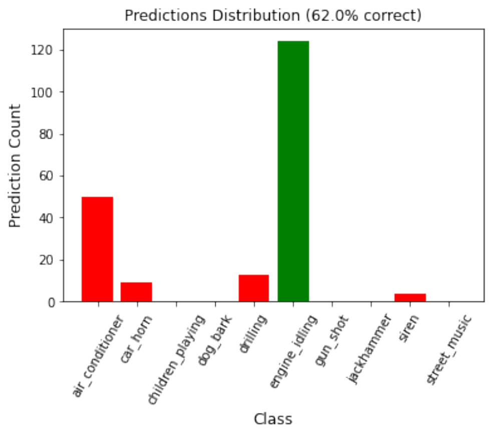


(b) Predicted Classes

Figure 3.14: Uncertainty Plots: Example 1: 20 MC Sample



(a) Predicted Probabilities



(b) Predicted Classes

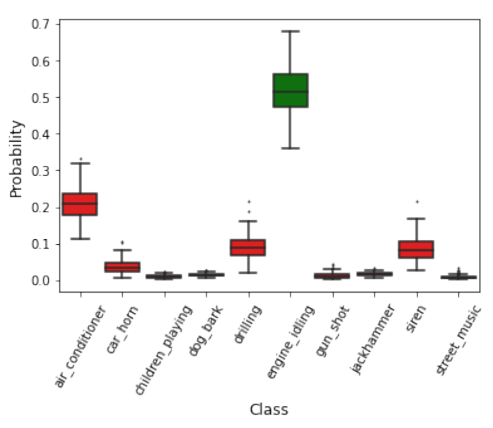
Figure 3.15: Uncertainty Plots: Example 2: 1 MC Sample

3.4.2 Test Example 2

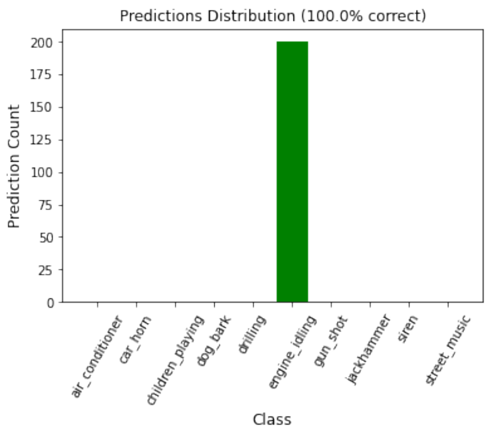
MC Samples: 1

While there is a different correct class, engine_idling, in this example, the plots for 1 MC sample in our second example are not dissimilar to that in example 1 in that the correct class is the most predicted class but it isn't predicted with great certainty. We see in Figure 3.15 (a) that the correct class has a large IQR, occupying a range from 0.3 to 0.82 roughly. The upper and lower extremes indicated by the "whiskers" of the plot span almost the entire possible range from 0 to 1. Three other incorrect classes are apportioned a significant probability with some frequency it appears, namely these are air_conditioner, drilling, and siren. Once again there are many outliers. All this suggests significant uncertainty in predictions, which is further exemplified by the bar chart in (b). The correct class is predicted only 62% of the time, while the aforementioned other incorrect classes apportioned significant probability are predicted for the majority of the rest of the time.

MC Samples: 20



(a) Predicted Probabilities



(b) Predicted Classes

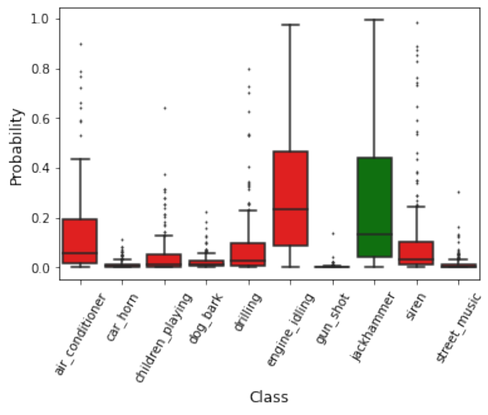
Figure 3.16: Uncertainty Plots: Example 2: 20 MC Sample

Similarly to example 1, we see a massive improvement in Figure 3.16 when using 20 MC samples as opposed to 1. Like before the IQRs are all narrowed. However, on this occasion, the range of the correct class is narrowed sufficiently to ensure it is predicted each time, as is exhibited by the bar chart in (b). In (a), the lowest predicted probability for the correct class engine_idling is roughly the same as the highest predicted for any other class.

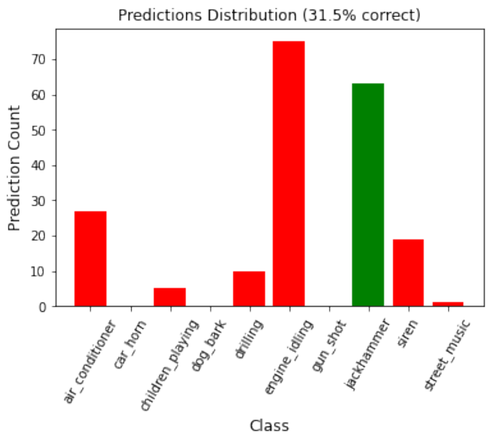
3.4.3 Test Example 3

MC Samples: 1

Example three is similar to example 1 in that the model is predicting more than one class. The difference is that in example 1, as we see from Figure 3.13, the correct class is the most predicted class. In Figure 3.17 (b) we see that the correct class jackhammer is predicted 31.5% of the time and engine_idling is predicted more often. In Figure 3.17 (a), we once again see relatively large IQRs. The distributions of probabilities for engine_idling and jackhammer are similar. Once again there s a large number of outliers.

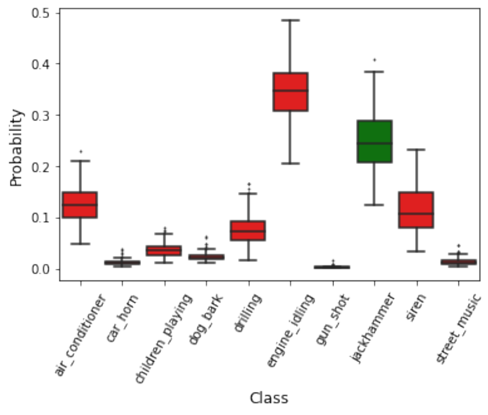


(a) Predicted Probabilities

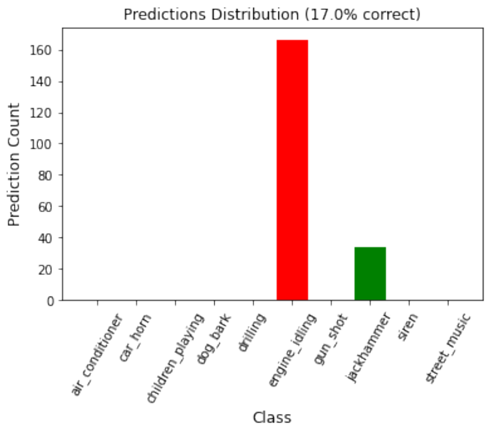


(b) Predicted Classes

Figure 3.17: Uncertainty Plots: Example 3: 1 MC Sample



(a) Predicted Probabilities



(b) Predicted Classes

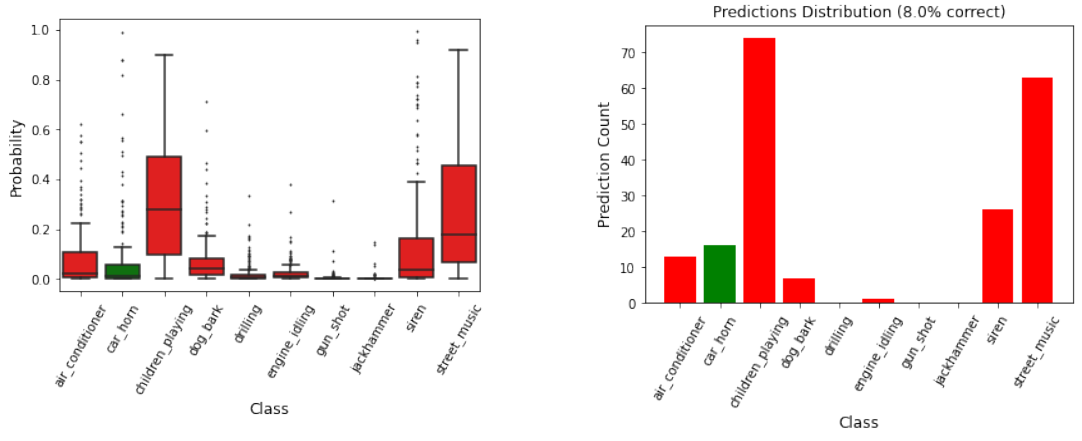
Figure 3.18: Uncertainty Plots: Example 3: 20 MC Sample

MC Samples: 20

Figure 3.18 presents an interesting contrast to Figure 3.15. In example 1 predcting with 1 MC samples, the predictions were mainly spread between two classes with the correct class being predicted slightly more. In example 3 with 1 MC samples the predictions were mainly spread between two classes with the correct class being predicted slightly less. In example 1, increasing the MC sample number to 20 increased the percent of the time the correct class was predicted. In example 3, the percent of the time the correct class was predicted decreased from 31.5% to 17%. This implies that increasing the MC sample number only improves performance for test examples that the model already predicts correctly more often than it predicts any one other class. Increasing the MC sample decreases the variance in the probability predictions for each class, making it less likely for the model to essentially "get lucky" and predict correctly when it is predicting a wide range of probability distributions.

3.4.4 Test Example 4

MC Samples: 1



(a) Predicted Probabilities

(b) Predicted Classes

Figure 3.19: Uncertainty Plots: Example 4: 1 MC Sample

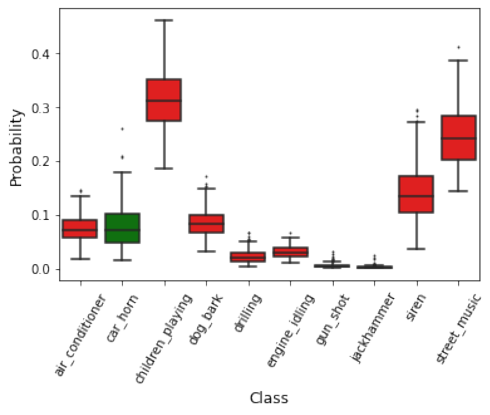
In the fourth and final example shown in Figure 3.19, we see predictions that are rarely correct with 1 MC sample. The model predominantly predicts children_playing and street_music when in fact the sound is car_horn. A high level of uncertainty is suggested by these probabilities.

MC Samples: 20

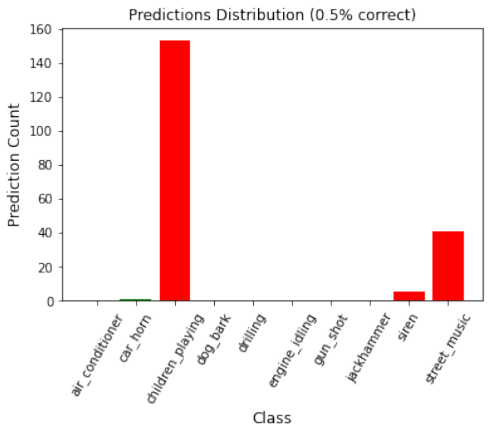
In Figure 3.20, we see that the correct class is predicted less often than in the 1 MC sample prediction. This is similar to example 3 in that regard. This reinforces the idea that increasing MC sample number only increases performance when the model has already been sufficiently trained. Like in the previously observed examples, the IQRs of all classes in Figure 3.20 (a) decreases in span and outliers are less frequent.

3.4.5 Uncertainty Plots Summary

All in all these plots can give us a good indication of how certain or uncertain the model is on a particular test example. Uncertainty can be interpreted through observing wider ranges in the probability apportioned to each class and many classes being predicted over the 200 forward passes. We have seen four examples with high apparent uncertainty here, particularly



(a) Predicted Probabilities



(b) Predicted Classes

Figure 3.20: Uncertainty Plots: Example 4: 20 MC Sample

when using only 1 MC sample. We observe all IQRs become narrower when moving to 20 MC samples but that this does not necessarily mean an increase in accuracy. This is specifically true when the model already has an inclination towards an incorrect class when using 1 MC sample.

3.5 Uncertainty Calculations

In the previous section we specified a particular example before making 200 predictions for it, then analysing the variety of the results. We can also measure the uncertainty on the full test set, as opposed to what we did in the last two parts where we picked out certain examples. In this section we quantify an estimation of uncertainty, both epistemic and aleatoric. We can break this section into three parts: predictive entropy, mutual information and aleatoric entropy. We use the same set of predictions for each part such that the predictive entropy equals the aleatoric entropy added to the mutual information (why this is the case is explained in the following sections).

3.5.1 Predictive Entropy

In information theory, the predictive entropy (sometimes referred to simply as entropy) of a random variable is the average level of surprise inherent in the variable's possible outcomes. It is a way to measure combined epistemic and aleatoric uncertainty in a model[62]. The predictive entropy of a predicted categorical probability distribution with n classes is defined as:

$$H = -\sum_{j=1}^{10} p_j \log_2(p_j) \tag{3.1}$$

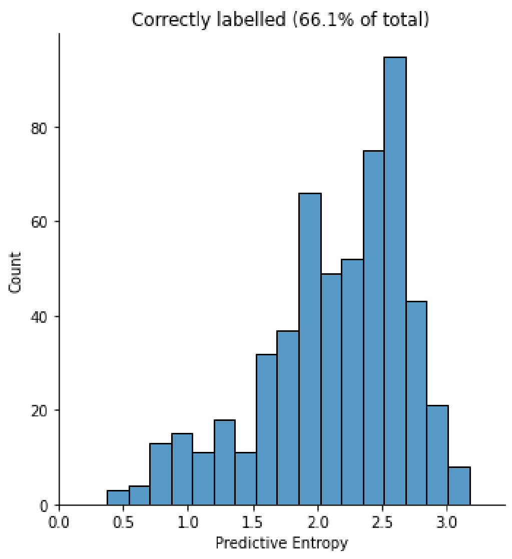
where p_j is the probability that the model assigns to class j. We calculate this separately for each test input. Therefore calculating over i test examples, we get i entropies. Base 2 gives the unit of bits (or "shannons", after being introduced by Claude Shannon in his 1948 paper[76]), while base e gives the "natural units" nat, and base 10 gives a unit called "dits", "bans", or "hartleys". We associate higher levels of uncertainty with higher predictive entropy scores. This is sometimes referred to as predictive entropy as it only uses the probabilities in our predictions in its calculation. A high predictive entropy score can indicate aleatoric uncertainty, in that the example is simply difficult to classify and this cannot be helped due to inherent stochasticity in the data. Alternatively, a high score can indicate high epistemic uncertainty, in that a probabilistic model has many possible explanations for the given input[29]. In this sense predictive entropy measures both aleatoric and epistemic uncertainty[62]. We will use this fact to calculate aleatoric entropy later. The predictive entropy in our model predictions can be approximated by the following formula:

$$H[p(y|x,D)] = H[\frac{1}{T}\sum_{i=1}^{T} p(y|x,w_i)] \tag{3.2}$$

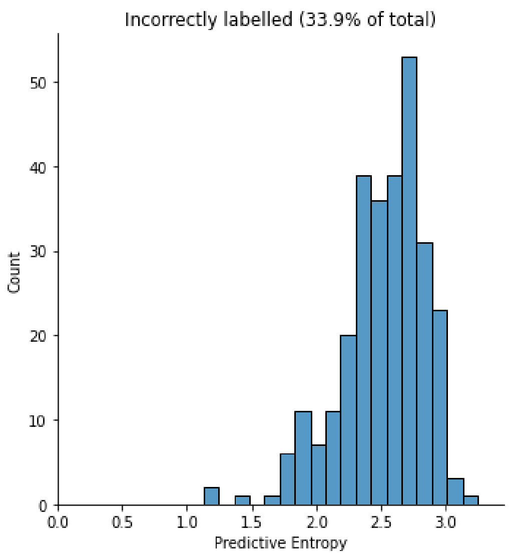
Hence we are essentially taking the predictive entropy of an average over T predictions. We can then use the formula in 3.1 to calculate the entropy. The following are plots we've generated based on our well performing model. We graph probability against entropy. We have separated the correctly labelled set from the incorrectly labelled in order to observe the difference.

We see from Figure 3.21 (a) and (b) that entropy is quite high among all predictions, both correct ones and incorrect. We would want the incorrect predictions to have a high entropy, as this suggests uncertainty when the model gets the prediction wrong. This is a desirable quality in a model. However, it would certainly be more desirable if our model was more confident about the correct predictions. The graph here in (a) is indicative that there is a high level of uncertainty in the model, as many predictions, correct and incorrect, have high uncertainty.

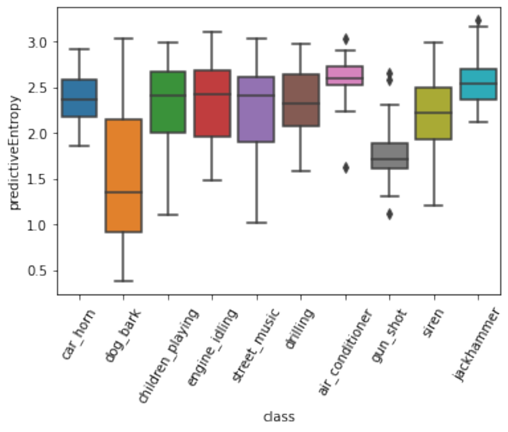
We can also plot the distributions of the entropies by class, which we do in Figure 3.21 (c). The Figure clearly shows that there is less uncertainty in the dog_bark examples, and to a lesser extent in the gun_shot examples. On the contrary jackhammer examples have slightly more uncertainty. This sort of information could be used in order to take more care with examples of these types, or alternatively to use data augmentation techniques in order to give the model more of these classes in training.



(a) Correctly Labelled Examples



(b) Incorrectly Labelled Examples



(c) Predictive Entropy by Class

Figure 3.21: Predictive Entropy Distributions

3.5.2 Mutual Information

The aforementioned predictive entropy isn't always a suitable measure of uncertainty however, as it does not distinguish between epistemic and aleatoric uncertainties [78]. Particularly, this is desirable in situations where we want to detect when an input is far from the region in the data space that the model's training data is located. In this way new examples completely different to the training data can be apportioned appropriate uncertainty. Furthermore, it may be beneficial to distinguish this from inputs near the data distribution with noisy labels. The information gain between data and the model parameters provides a measure of epistemic uncertainty for this purpose. This can be given by the mutual information (MI) formula. We give the generic equation for the mutual information between two variables here:

$$\begin{aligned} I(X,Y) &= H[P(X)] - E_{P(y)}H[p(X|Y)] \\ &= H[P(Y)] - E_{P(x)}H[p(Y|X)] \end{aligned} \tag{3.3}$$

The amount of information we would gain about the model parameters if we were to receive a label y for a new point x, given the dataset D is therefore given by:

$$I(w,y|x,D) = H[p(y|x,D)] - E_{P(w|D)}H[p(y|x,w)] \tag{3.4}$$

The measure intuitively classifies as uncertain any any input for which knowing the label would increase information. This would mean a high MI score. Vice versa, if the parameters for a particular input are well defined then knowing the label would result in little or no gain in information and thus a low MI score. This would imply little uncertainty. The first term in equation 3.4 is the predictive entropy, the same as the measure described in the previous section 3.5.2. In the second term we see the predictive entropy $H[P(y|x,D)]$ once again, but this time we take the mean of this over the posterior distribution $p(w|D)$. We refer to this as the expected entropy. The integral required to calculate this term analytically is computationally intractable for any complex networks, so we approximate using the following equations:

$$\begin{aligned} p(y|x,D) &\approx \frac{1}{T}\sum_{i=1}^{T} p(y|x,w_i) \\ &:= p_{MC}(y|x) \end{aligned} \tag{3.5}$$

———

$$H[p(y|x,D)] \approx H[p_{MC}(y|x)] \tag{3.6}$$

———

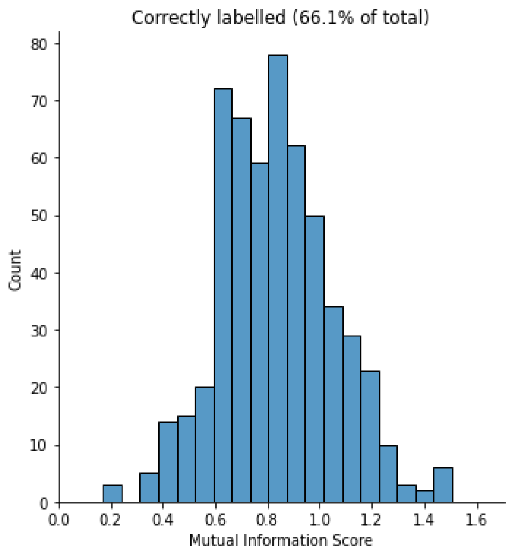
$$E_{p(w|D)}[f^{w}(x)] \approx \frac{1}{T}\sum_{i=1}^{T} f^{w_i}(x) \tag{3.7}$$

Equation 3.5 refers to how we approximate the distribution of the prediction probabilities using the neural network. The w_i refers to one sample from the posterior distribution on the weights. Thus the term $p_{MC}(y|x)$ can be seen as the models prediction of input x using T MC samples, as described earlier. Equation 3.6 says we can approximate the entropy on $p(y|x,D)$

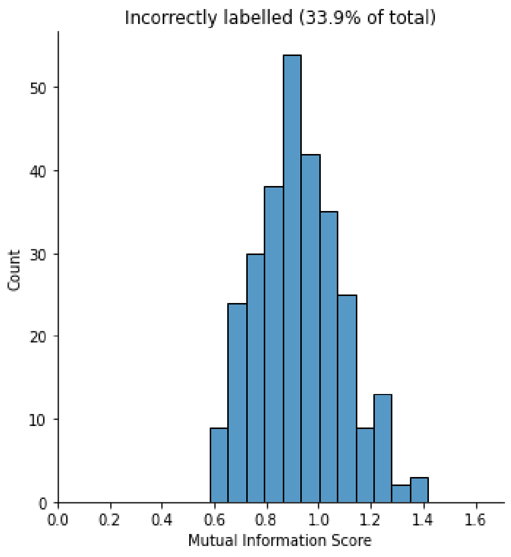
by using $p_{MC}(y|x)$ from the previous equation. Equation 3.7 is the MC sample estimator we also referred to in chapter 1. We can put these together to find the following computationally tractable equation for mutual information by subbing them into equation 3.4:

$$I(w, y|x, D) \approx H[p_{MC}(y|x)] - \frac{1}{T}\sum_{i=1}^{T} H[p(y|x, w_i)] \tag{3.8}$$

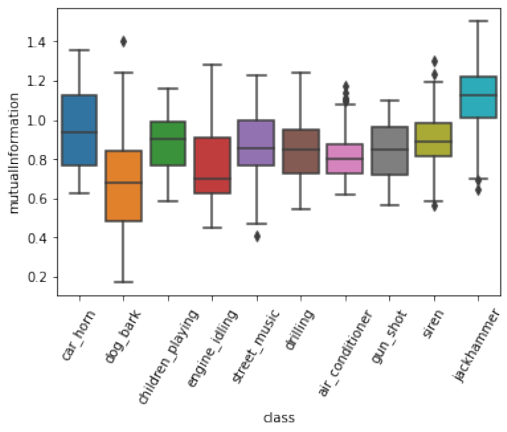
The first term is the entropy of T probability predictions averaged. The second term is the average of T entropies as calculated on T predicted probabilities. The second taken away from the first gives us an approximation of the MI and thus the epistemic uncertainty in the model. We plot the distributions of the mutual information scores in Figure 3.22. In (a) and (b) we see the distribution of the mutual information scores for correctly and incorrectly predicted test inputs respectively. In (c), we see a boxplot of mutual information scores broken down by class.



(a) Correctly Labelled Examples



(b) Incorrectly Labelled Examples



(c) Mutual Information by Class

Figure 3.22: Mutual Information Distributions

For correct classes, the average mutual information score was 0.80551. For incorrect classes, 0.9604. So we can conclude that there is generally more epistemic uncertainty associated with the test examples that were incorrectly predicted, which is what we want. However, there is still a high level of uncertainty prevalent in the correctly labelled test examples. This on the contrary, is undesirable. Why the model is has a high level of uncertainty is another question which is difficult to answer, with the root cause being difficult to pinpoint. Plots like these are impossible with a regular deep neural network as all forward passes result in the same set of prediction probabilities for the set of inputs. They show how we can quantify and visualise uncertainty in our model, both epistemic with mutual information and a mixture of epistemic and aleatoric with predictive entropy. They provide us with enriched results in that the predicted probabilities are not all we can find from a BNN.

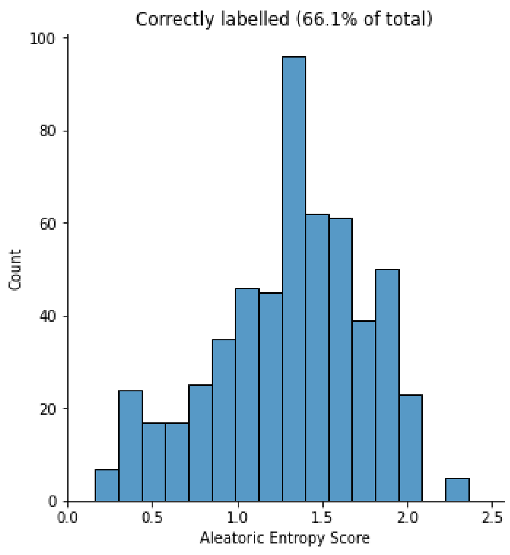
Once again, we can plot the class by class mutual information. Similarly to Figure 3.21, dog_bark has the least associated uncertainty while jackhammer has the highest. The rest of the mutual informations have IQRs in with similar medians and slightly varying spans. This suggests uncertainty is similar in general throughout these classes.

3.5.3 Aleatoric Entropy

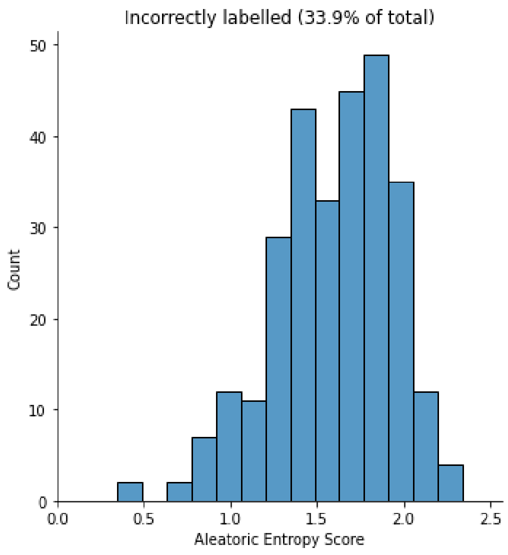
Aleatoric entropy can also be approximated (as described in this reference [62]) by simply subtracting the mutual information $I(w, y|x, D)$ from predictive entropy $H[p(y|x, D)]$. The predictive entropy representing both epistemic and aleatoric uncertainty is additive in this sense; it is found by adding aleatoric and epistemic uncertainty estimates. This results in the second term on the right hand side of equation 3.2: $E_{P(w|D)}H[p(y|x, w)]$. This can be approximated by the same methods described above. Its approximated form is given by the formula:

$$E_{P(w|D)}H[p(y|x, w)] = \frac{1}{T}\sum_{i=1}^{T} H[p(y|x, w_i)] \tag{3.9}$$

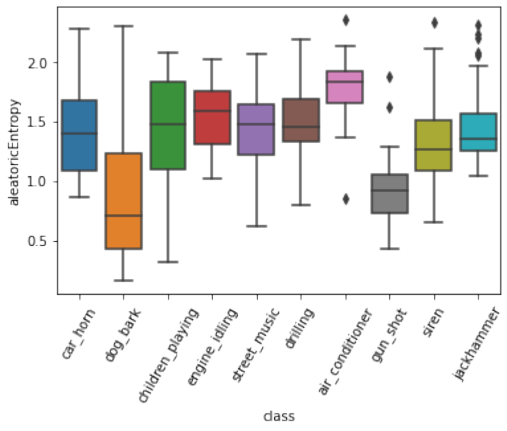
In Figure 3.23 we plot the distributions of the aleatoric entropies. We see similar results to that of predictive entropy and mutual information. The correct predictions are slightly less uncertain than the incorrect predictions, which is positive. Similar to in the previous figures though, the correct predictions still have a high uncertainty. Ideally the model would be more distinctly uncertain about the incorrect predictions than the correct predictions.



(a) Correctly Labelled Examples



(b) Incorrectly Labelled Examples



(c) Aleatoric Entropy by Class

Figure 3.23: Aleatoric Entropy Distributions

3.5.4 Uncertainty Calculations Summary

Firstly we note the similarity between all in figures 3.21-3.23 (c). The class dog_bark generally has the lowest uncertainty associated, while jackhammer and air_conditioner have the most overall uncertainty. The difference in these latter two however, is that the jackhammers uncertainty is more epistemic, while the air_conditioner uncertainty is more aleatoric. This suggests a different root cause. It suggests the air_conditioner inherently has more stochasticity, while the uncertainty of the jackhammer is originating in the model somewhere. Where exactly is another question, which we don't have scope to go into for the purpose of this thesis.

In the following table, we give the mean uncertainty scores for incorrect and correctly labelled test examples.

Mean:	**Aleatoric Entropy**	**Mutual Information**	**Predictive Entropy**
Correct	1.3002	0.8312	2.1289
Incorrect	1.5887	0.9265	2.5153

We clearly see that generally the uncertainty in the incorrect predictions is higher.

All in all these uncertainty estimations give us enriched information on data. They allow us to break down uncertainty into aleatoric and epistemic across an entire dataset. They can be used to give more interpretability on results, which is particularly useful in mission critical tasks. They can be used to reject certain predictions in this sense, as we can set an uncertainty threshold over which we do not accept predictions. Alternatively, we can reject a percentage of the test data with the highest uncertainty. This is what we will do in the next section.

3.6 Rejection-Classification

Having described uncertainty estimations in the previous section, we have shown how it can be used as a score of how uncertain a prediction is. We can use these to create rejection classification plots [85]. A rejection classification graph plots the test accuracy against the percentage of the test predictions thrown away, where we throw away the predictions with the highest uncertainty scores first, and presume these to be incorrect. We then calculate accuracy based on the remaining predictions. We should expect the accuracy to change increase or decrease as the percentage of the test set we throw away increases. If the model is trained adequately and calculating uncertainty accurately, then the first predictions to be thrown away should wrong as these are the ones we would ideally have the greatest uncertainty for. On the contrary, if the model was somehow attributing higher uncertainty to the correct predictions than. incorrect, then accuracy would go down initially.

We see the rejection classification plots in Figure 3.24 where we have used all three uncertainty estimations. Generally, as the percentage thrown away increases, so too does the accuracy. This is a desirable characteristic in our model and shows that it is uncertain on mostly incorrect predictions. We also plot the theoretical maximum, which would occur if all the incorrect predictions were thrown away first before any correctly predicted examples. This of course would be exactly what we want. It's obvious from the plots that our model is some way off perfect. This is related to Figures 3.21-3.23 in the previous section, where we observed that while generally the incorrectly predicted examples had higher uncertainty, there wasn't as much of a distinction between these and the correct prediction uncertainties as we would

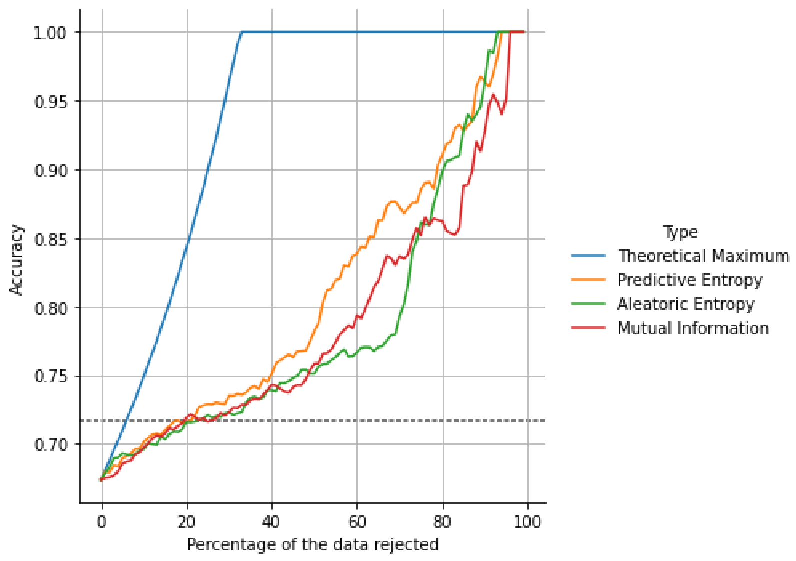


Figure 3.24: Rejection Classification Graph

like. Generally however, the plots ascend reasonably quickly in the graph. We graph a broken horizontal line to represent the deterministic M18 test accuracy of 71.68%. The following table summarises what percentage of the test set we need to throw away in order to match the deterministic M18 test recorded test accuracy.

	Percentage (%)
Aleatoric Entropy	23
Mutual Information	20
Predictive Entropy	18
Theoretical Maximum	8

The fact that predictive entropy is an estimate of both aleatoric and epistemic uncertainty, and that it gives us the best rejection classification plot in that it is closest to the theoretical maximum, suggests the uncertainty in our predictions is best described as a mixture of aleatoric and epistemic. Graphs of these sort generally indicate a models ability to detect difficulty in classifying examples[85], where the closer the plots are to the theoretical maximum the better. This would suggest the model knows what it does not know. By this logic our model measures uncertainty relatively well, while there is definitely some room for improvement.

Conclusion

4.1 Summary of Thesis Achievements

We introduced this thesis by presenting the task of ESC and where it is found in real-life mission-critical scenarios. We specifically referenced use cases in which a measure of uncertainty would be particularly beneficial. Little research could be found specifically on the use of BNNs specifically in the ESC task. For this reason our literary review approached both concepts separately. We summarised some previous work on the ESC task (specifically with the Urban Sounds 8K and ESC-50 datasets). We also gave reference to uncertainty quantification with BNNs and how it has been used in other (non-ESC) classification tasks. We formulated motivation as to why BNNs could provide a useful tool in providing uncertainty calculations in such tasks before describing the theory behind such methods. This provided the backdrop to what we wanted to investigate in this thesis.

Using knowledge of previous work in the field of ESC from the Section "Prior Work", we selected the M18 model as our starting point for implementing a BNN. The goal was to build a BNN, named M18-P, with the same architecture as the deterministic M18 and see whether we could match performance while making it possible to calculate uncertainty. We provided analysis on five key hyperparameters in the model. We found that increasing the MC sample number used in test predictions greatly increased accuracy. On closer inspection however, we gained understanding on which specific predictions benefited from a higher MC sample number. It was found that, over 200 predictions for one input, an increased MC sample number only resulted in an increased number of correct predictions when the correct class was already the most predicted class. This also meant that the model in fact performed worse when the MC sample number was increased for test examples where an incorrect class was the most frequently predicted initially. This would suggest that increased MC sample number would have even better performance gains in a model that achieves greater accuracy for lower

MC sample numbers. Due to time constraints however we could not test hypothesis. We found that the batch size had a huge effect on the performance and attributed this to the increased stochasticity in BNNs as opposed to deterministic models. We found that the KL divergence approximation sample number didn't have a great effect on performance which was somewhat surprising as intuition might suggest a better approximation on the distributions would mean more accurate results. The final two parameters we analysed were posterior and prior distributions. We found that priors with heavier tails generally performed better which was consistent with past research. The inverse was true for the posteriors where we observed distributions with less tailweight seemed to perform better.

M18-P achieved comparable, but definitely slightly worse, results to M18. We were able to calculate aleatoric and epistemic uncertainties as well as the combined values. We found that the model was slightly more uncertain on incorrect predictions than correct ones. This is a desirable trait in a model. We leveraged these uncertainty calculations to generate rejection classification plots and found that we could replicate the best test accuracy of the deterministic model by rejecting 18% of predictions based on predictive entropy.

We have shown in this thesis advantages and disadvantages to using BNNs in an ESC task. We can conclude that there is extra work required in implementing a BNN compared to a regular neural network. Much of the time spent on this thesis was spent in finding the right mix of hyperparameters. The need for the prior and posterior to be defined can be can be useful in situations where information is known about the weight distributions before training. If not, it can take some time to find the right selection. Further to this, we saw an increase in train and test times. We also seen a slight drop off in performance. The advantage of BNNs is the ability to calculate aleatoric and epistemic uncertainty. In this thesis we showed how this can be used in rejection classification. Whether or not a BNN is suitable for any given task will depend on whether these trade-offs are acceptable and how important it is to calculate uncertainty.

4.2 Further Research

A potential criticism of this thesis that leads into this section on further research is hyperparameter selection. While we attempted to be as exhaustive as possible within the time constraints of the thesis, different methods of hyperparameter selection could be applied more effectively. For instance, the use of random search or Bayesian search for hyperparameter selection. These could have been more time effective than the method we employed and could have resulted in better performance. Following the analysis of hyperparameters in this paper we could now choose better ranges within which to search. In the case of the priors and posteriors, knowing which distributions generally performed better would enable us to narrow the scope of distributions considered.

Further work would be required to examine the usefulness of BNNs in ESC tasks generally. We attempted to include a different model which uses both the Urban Sounds 8K dataset and the ESC-50 dataset[90]. This model extracted log-Mel spectrograms from the audio files and used mix-up data augmentation which would have given us a different perspective on the suitability of BNNs for ESC. Unfortunately, with time and thesis length constraints we decided to focus our analysis on M18-P. Incorporating another dataset would benefit this cause also.

We also experimented with the possibility of using a mixture of deterministic and probabilistic layers in the network. We dropped off probabilistic layers one by one and replaced them with deterministic ones. We recorded statistics for each new model. Similarly to the last point, we were unable to finish this due to the constraints of the thesis. This could be a potential way to bridge the gap in performance between M18 and M18-P while still maintaining a method to calculate uncertainty in predictions.

Bibliography

[1] Dharmesh M Agrawal, Hardik B Sailor, Meet H Soni, and Hemant A Patil. Novel teo-based gammatone features for environmental sound classification. In *2017 25th European Signal Processing Conference (EUSIPCO)*, pages 1809–1813. IEEE, 2017.

[2] Yasser Alsouda, Sabri Pllana, and Arianit Kurti. A machine learning driven iot solution for noise classification in smart cities, 2018.

[3] Alexander Amini, Ava Soleimany, Sertac Karaman, and Daniela Rus, 12 2017.

[4] Tom Auld, Andrew W Moore, and Stephen F Gull. Bayesian neural networks for internet traffic classification. *IEEE Transactions on neural networks*, 18(1):223–239, 2007.

[5] Oleg Yu Bakhteev and Vadim V Strijov. Comprehensive analysis of gradient-based hyper-parameter optimization algorithms. *Annals of Operations Research*, 289(1):51–65, 2020.

[6] Yoshua Bengio. *Practical Recommendations for Gradient-Based Training of Deep Architectures*, pages 437–478. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[7] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[8] José M Bernardo and Adrian FM Smith. *Bayesian theory*, volume 405. John Wiley & Sons, 2009.

[9] David M Blei and Michael I Jordan. Variational inference for dirichlet process mixtures. *Bayesian analysis*, 1(1):121–143, 2006.

[10] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, Apr 2017.

[11] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. *Weight Uncertainty in Neural Networks.* Proceedings of the 32nd International Conference on Machine Learning, PMLR 37:1613-1622, 2015, 2015.

[12] Erik Bochinski, Tobias Senst, and Thomas Sikora. Hyper-parameter optimization for convolutional neural network committees based on evolutionary algorithms. In *2017 IEEE international conference on image processing (ICIP)*, pages 3924–3928. IEEE, 2017.

[13] Michael Braun and Jon McAuliffe. Variational inference for large-scale models of discrete choice. *Journal of the American Statistical Association*, 105(489):324–335, Mar 2010.

[14] Lex Brown, Jian Kang, and Truls Gjestland. Towards standardization in soundscape preference assessment. *Applied Acoustics - APPL ACOUST*, 72:387–392, 05 2011.

[15] Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. *ArXiv*, abs/1712.09665, 2017.

[16] Wray L. Buntine and Andreas S. Weigend. Bayesian back-propagation. *Complex Syst.*, 5, 1991.

[17] Jiwoong Choi, Dayoung Chun, Hyun Kim, and Hyuk-Jae Lee. Gaussian yolov3: An accurate and fast object detector using localization uncertainty for autonomous driving, 2019.

[18] Selina M. Chu, Shrikanth S. Narayanan, and C.-C. Jay Kuo. Environmental sound recognition with time–frequency audio features. *IEEE Transactions on Audio, Speech, and Language Processing*, 17:1142–1158, 2009.

[19] Giuseppe Ciaburro and Gino Iannace. Improving smart cities safety using sound events detection based on deep neural network algorithms. *Informatics*, 7(3), 2020.

[20] Yandre MG Costa, Luiz S Oliveira, and Carlos N Silla Jr. An evaluation of convolutional neural networks for music classification using spectrograms. *Applied soft computing*, 52:28–38, 2017.

[21] Marco Crocco, Marco Cristani, Andrea Trucco, and Vittorio Murino. Audio surveillance: A systematic review. *ACM Comput. Surv.*, 48(4), February 2016.

[22] Wei Dai, Chia Dai, Shuhui Qu, Juncheng Li, and Samarjit Das. *Very Deep Convolutional Neural Networks for Raw Wave Forms.* International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2017, 2016.

[23] Muqing Deng, Tingting Meng, Jiuwen Cao, Shimin Wang, Jing Zhang, and Huijie Fan. Heart sound classification based on improved mfcc features and convolutional recurrent neural networks. *Neural Networks*, 130:22–32, 2020.

[24] Joshua V. Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A. Saurous. Tensorflow distributions, 2017.

[25] Lorcan Donnellan. Github location for code to accompany thesis"uncertainty and bayesian neural networks in environmental sound classification", 2021. https://github.com/lorcd/imperialThesis.git.

[26] Michael W. Dusenberry, Ghassen Jerfel, Yeming Wen, Yi-An Ma, Jasper Snoek, Katherine Heller, Balaji Lakshminarayanan, and Dustin Tran. Efficient and scalable bayesian neural nets with rank-1 factors, 2020.

[27] Stefan Eggenreich, Christian Payer, Martin Urschler, and Darko Štern. Variational inference and bayesian cnns for uncertainty estimation in multi-factorial bone age prediction, 2020.

[28] Di Feng, Lars Rosenbaum, and Klaus Dietmayer. Towards safe autonomous driving: Capture uncertainty in the deep neural network for lidar 3d vehicle detection, 2018.

[29] Angelos Filos, Sebastian Farquhar, Aidan N. Gomez, Tim G. J. Rudner, Zachary Kenton, Lewis Smith, Milad Alizadeh, Arnoud de Kroon, and Yarin Gal. A systematic comparison of bayesian deep learning robustness in diabetic retinopathy tasks, 2019.

[30] Eduardo Fonseca, Jordi Pons Puig, Xavier Favory, Frederic Font Corbera, Dmitry Bogdanov, Andres Ferraro, Sergio Oramas, Alastair Porter, and Xavier Serra. Freesound datasets: a platform for the creation of open audio datasets. In *Hu X, Cunningham SJ, Turnbull D, Duan Z, editors. Proceedings of the 18th ISMIR Conference; 2017 oct 23-27; Suzhou, China.[Canada]: International Society for Music Information Retrieval; 2017. p. 486-93.* International Society for Music Information Retrieval (ISMIR), 2017.

[31] Vincent Fortuin, Adrià Garriga-Alonso, Mark van der Wilk, and Laurence Aitchison. Bnpriors: A library for bayesian neural network inference with different prior distributions. *Software Impacts*, 9:100079, 2021.

[32] Vincent Fortuin, Adrià Garriga-Alonso, Florian Wenzel, Gunnar Rätsch, Richard Turner, Mark van der Wilk, and Laurence Aitchison. Bayesian neural network priors revisited, 2021.

[33] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 2004.

[34] Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Humt, Jianxiang Feng, Anna Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, Muhammad Shahzad, Wen Yang, Richard Bamler, and Xiao Xiang Zhu. A survey of uncertainty in deep neural networks, 2021.

[35] Andrew Gelman, John B Carlin, Hal S Stern, and Donald B Rubin. *Bayesian data analysis*. Chapman and Hall/CRC, 1995.

[36] Ryan Giordano, Tamara Broderick, and Michael Jordan. Linear response methods for accurate covariance estimates from mean field variational bayes, 2015.

[37] Abhijit Guha Roy, Sailesh Conjeti, Nassir Navab, and Christian Wachinger. Bayesian quicknat: Model uncertainty in deep whole-brain segmentation for structure-wise quality control. *NeuroImage*, 195, 03 2019.

[38] Jose Miguel Hernandez-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1861–1869, Lille, France, 07–09 Jul 2015. PMLR.

[39] Yedid Hoshen, Ron J. Weiss, and Kevin W. Wilson. Speech acoustic modeling from raw multichannel waveforms. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4624–4628, 2015.

[40] Philip D. Howard. The effect of sample heterogeneity and risk categorization on area under the curve predictive validity metrics. *Criminal Justice and Behavior*, 44(1):103–120, 2017.

[41] DH Hubel and TN Wiesel. 8. receptive fields of single neurones in the cat's striate cortex. In *Brain Physiology and Psychology*, pages 129–150. University of California Press, 2020.

[42] Kurt Jacobson, Vidhya Murali, Edward Newett, Brian Whitman, and Romain Yon. Music personalization at spotify. In *Proceedings of the 10th ACM Conference on Recommender*

Systems, RecSys '16, page 373, New York, NY, USA, 2016. Association for Computing Machinery.

[43] Toru Kamekawa and Atsushi Marui. Physical factors and spatial impressions of surround sound recording. In *2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pages 580–585, 2012.

[44] Jaehun Kim. Urban sound tagging using multi-channel audio feature with convolutional neural networks. *Detection and Classification of Acoustic Scenes and Events 2020*, 2020.

[45] Igor Kononenko. Bayesian neural networks. *Biological Cybernetics*, 61(5):361–370, 1989.

[46] Alp Kucukelbir, Rajesh Ranganath, Andrew Gelman, and David Blei. Automatic variational inference in stan. 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.

[47] 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.

[48] Yongchan Kwon, Joong-Ho Won, Beom Joon Kim, and Myunghee Cho Paik. Uncertainty quantification using bayesian neural networks in classification: Application to biomedical image segmentation. *Computational Statistics & Data Analysis*, 142:106816, 2020.

[49] Jouko Lampinen and Aki Vehtari. Bayesian approach for neural networks—review and case studies. *Neural networks*, 14(3):257–274, 2001.

[50] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[51] Chang-Hsing Lee, Chih-Hsun Chou, Chin-Chuan Han, and Ren-Zhuang Huang. Automatic recognition of animal vocalizations using averaged mfcc and linear discriminant analysis. *pattern recognition letters*, 27(2):93–101, 2006.

[52] Christos Louizos and Max Welling. Multiplicative normalizing flows for variational bayesian neural networks. In *International Conference on Machine Learning*, pages 2218–2227. PMLR, 2017.

[53] Tao Lv, He yong Zhang, and Chun hui Yan. Double mode surveillance system based on remote audio/video signals acquisition. *Applied Acoustics*, 129:316–321, 2018.

[54] David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

[55] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[56] Vikram Mullachery, Aniruddh Khera, and Amir Husain. Bayesian neural networks. *arXiv preprint arXiv:1801.07710*, 2018.

[57] Zohaib Mushtaq and Shun-Feng Su. Efficient classification of environmental sounds through multiple features aggregation and data enhancement techniques for spectrogram images. *Computer and Engineering Science and Symmetry/Asymmetry*, 2020.

[58] Tanya Nair, Doina Precup, Douglas L. Arnold, and Tal Arbel. Exploring uncertainty measures in deep networks for multiple sclerosis lesion detection and segmentation, 2018.

[59] Loris Nanni, Gianluca Maguolo, Sheryl Brahnam, and Michelangelo Paci. An ensemble of convolutional neural networks for audio classification. *Applied Sciences*, 11(13), 2021.

[60] Radford Neal. Bayesian learning via stochastic dynamics. In S. Hanson, J. Cowan, and C. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1993.

[61] Karen Haoarchive page. Ai is sending people to jail—and getting it wrong. *MIT Technology Review*, 2019.

[62] Buu Phan, Samin Khan, Rick Salay, and Krzysztof Czarnecki. Bayesian uncertainty quantification with synthetic data. In Alexander Romanovsky, Elena Troubitsyna, Ilir Gashi, Erwin Schoitsch, and Friedemann Bitsch, editors, *Computer Safety, Reliability, and Security*, pages 378–390, Cham, 2019. Springer International Publishing.

[63] K.J Piczak. *Environmental sound classification with convolutional neural networks*. IEEE, 2015.

[64] K.J Piczak. *ESC: Dataset for environmental sound classification*. MM '15: Proceedings of the 23rd ACM international conference on MultimediaOctober 2015 Pages 1015–1018, 2015.

[65] Arya A Pourzanjani, Richard M Jiang, and Linda R Petzold. Improving the identifiability of neural networks for bayesian inference. In *NIPS Workshop on Bayesian Deep Learning*, volume 4, page 29, 2017.

[66] Asma Rabaoui, Manuel Davy, StÉphane Rossignol, and Noureddine Ellouze. Using one-class svms and wavelets for audio surveillance. *IEEE Transactions on Information Forensics and Security*, 3(4):763–775, 2008.

[67] Manon Raimbault and Danièle Dubois. Urban soundscapes: Experiences and knowledge. *Cities*, 22:339–350, 10 2005.

[68] Jacob C. Reinhold, Yufan He, Shizhong Han, Yunqiang Chen, Dashan Gao, Junghoon Lee, Jerry L. Prince, and Aaron Carass. Validating uncertainty in medical image translation, 2020.

[69] Christian P. Robert and George Casella. Monte carlo statistical methods. In *Springer Texts in Statistics*, 2004.

[70] Frank Rosenblatt. Perceptron simulation experiments. *Proceedings of the IRE*, 48(3):301–309, 1960.

[71] Abhijit Guha Roy, Sailesh Conjeti, and Christian Wachinger Nassir Navab. Bayesian quicknat: Model uncertainty in deep whole-brain segmentation for structure-wise quality control. *NeuroImage; Volume 195, 15 July 2019, Pages 11-22*, 2018.

[72] J. Salamon, C. Jacoby, and J. P. Bello. A dataset and taxonomy for urban sound research. In *22nd ACM International Conference on Multimedia (ACM-MM'14)*, pages 1041–1044, Orlando, FL, USA, Nov. 2014.

[73] Justin Salamon and Juan Pablo Bello. Deep convolutional neural networks and data augmentation for environmental sound classification. *CoRR*, abs/1608.04363, 2016.

[74] Loza Bekalo Sappa, Idowu Paul Okuwobi, Mingchao Li, Yuhan Zhang, Sha Xie, Songtao Yuan, and Qiang Chen. Retfluidnet: Retinal fluid segmentation for sd-oct images using convolutional neural network. *Journal of Digital Imaging*, pages 1–14, 2021.

[75] ROMAIN SERIZEL and DIEGO GIULIANI. Deep-neural network approaches for speech recognition with heterogeneous groups of speakers including children. *Natural Language Engineering*, 23(3):325–350, 2017.

[76] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[77] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. *CCS '16: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[78] Lewis Smith and Yarin Gal. Understanding measures of uncertainty for adversarial example detection, 2018.

[79] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

[80] Hans Henrik Thodberg. A review of bayesian neural networks with an application to near infrared spectroscopy. *IEEE transactions on Neural Networks*, 7(1):56–72, 1996.

[81] Christopher Thomas. An introduction to convolutional neural networks, 2019. Online resource.

[82] Tishby, Levin, and Solla. Consistent inference of probabilities in layered networks: predictions and generalizations. In *International 1989 Joint Conference on Neural Networks*, pages 403–409 vol.2, 1989.

[83] User-uploaded. Free sound database. www.freesound.org.

[84] Michel Vacher, Dan Istrate, Laurent Besacier, Jean-François Serignat, and Eric Castelli. Sound detection and classification for medical telesurvey. In *2nd Conference on Biomedical Engineering*, pages 395–398, 2004.

[85] Joost van Amersfoort, Lewis Smith, Yee Whye Teh, and Yarin Gal. Uncertainty estimation using a single deep deterministic neural network. *ICML 2020*, 2020.

[86] Kevin Webster. *Probabilistic Deep Learning with TensorFlow 2*, 2020. Available at https://www.coursera.org/learn/probabilistic-deep-learning-with-tensorflow2.

[87] Kevin Webster. Imperial College London, Department of Mathematics, Data Science Module: MATH97097, Lecture Notes: Neural Networks-Convolutional Neural Networks, 2021. Lecture notes.

[88] Felix Weninger and Björn Schuller. Audio recognition in the wild: Static and dynamic classification on a real-world database of animal vocalizations. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 337–340. IEEE, 2011.

[89] Zhichao Zhang, Shugong Xu, Shan Cao, and Shunqing Zhang. Deep convolutional neural network with mixup for environmental sound classification. In Jian-Huang Lai, Cheng-Lin Liu, Xilin Chen, Jie Zhou, Tieniu Tan, Nanning Zheng, and Hongbin Zha, editors, *Pattern Recognition and Computer Vision*, pages 356–367, Cham, 2018. Springer International Publishing.

[90] Zhichao Zhang, Shugong Xu, Shan Cao, and Shunqing Zhang. Deep convolutional neural network with mixup for environmental sound classification, 2018.

[91] Victor Zhou. Machine learning for beginners: An introduction to neural networks, 2019. Online resource.

List of Figures