Imperial College London

MSc Statistics Dissertation

Imperial College London

Department of Mathematics

Bayesian Neural Network Audio Classifiers

Author:
Jack Robert Norrie

Supervisor:
Dr. Kevin Webster

Date: September 17, 2021

Acknowledgements

I would like to thank my supervisor, Dr Kevin Webster, for his continued support throughout the course of this research project. His expertise within the field of deep learning and proficiency in using the frameworks associated with this field, specifically TensorFlow Probability, allowed for him to give me guidance throughout this project that was essential for this project to reach the developed stage that it did. I would also like to thank Mr Andy Thomas for the technical support he provided me throughout the course of this project, he was very quick to respond to any queries I had surrounding the GPU cluster that I was using for the computation associated with this project. Furthermore, I would like to thank Dr Dean Bodenham and Dr Marina Evangelou for the arrangements they made to ensure that this project reached its full potential. Finally, I would like to thank my parents, Dr Gillian Norrie and Professor John Norrie, for their support throughout my university studies, without which none of this would be possible.

Abstract

Audio classification problems are characteristically challenging machine learning problems. They typically operate within limited data regimes, due to the laborious nature of labelling audio data. This paired with the fact that audio data is typically very high dimensional and the result is a dataset that explores a very small fraction of the input space. In relation to artificial neural networks this leads to two main consequences, the first being that audio classifiers tend to be at an increased risk of overfitting and the second being that artificial neural networks fit to such datasets are typically under-specified. Under-specification refers to the phenomenon whereby there are many hypotheses/parameter values that are consistent with the data, selection of one parameter over another could be considered rather arbitrary, as such the data has not been able to specify a clear optimum hypothesis. The objective of this report was to investigate Bayesian Neural Networks (BNNs) as a potential remedy for this audio classification artificial neural network under-specification problem. The idea being that a Bayesian neural network's estimated Bayesian model average could potentially exploit under specification by essentially taking an ensemble of well performing and, ideally, functionally different hypotheses. The hypotheses being functionally different would potentially allow for one hypothesis' poor predictions to be mitigated through aggregation with another well performing but functionally different hypothesis. A related approach is that of deep ensembles, whereby a neural network is trained M times with different random initialisations. The converged upon solutions/hypotheses are all functionally different and high performing, hence aggregation of such solutions can lead to improvements in performance. The literature to this date suggests that the deep ensemble approach is a more effective approach, however it is also considerably more computationally expensive.

The "ESC-50: Dataset for Environmental Sound Classification" dataset was used to investigate the objective of this work. First an effective baseline Convolutional Neural Network (CNN) architecture was determined, this achieved a 5-fold cross validation accuracy of $(72.51 \pm 1.53)\%$. Unfortunately, the comparable BNN architecture only achieved a classification accuracy of $(70.60 \pm 2.65)\%$. Furthermore, ensembles consisting of 5 of these networks each were evaluated on the ESC-50 datasets, the CNN and BNN ensemble networks achieved 5-fold cross validation accuracies of $(76.00 \pm 1.058)\%$ and $(73.60 \pm 2.04)\%$ respectively. The fold-wise validation accuracies forming these 5 fold cross validation accuracy estimates were then used in a one sided paired t-test, testing the null hypothesis that the BNN networks would perform as well if not worse than their comparable CNN networks. Unfortunately, the results from both of the pairings BNN/CNN and BNN ensemble/CNN ensemble failed to reject the null hypothesis, this being the case even with the reduction in power induced by the correlation between fold-wise validation accuracy estimates.

Contents

Notation

General Mathematics

$\bigcup$, $\bigcap$	Set Union, Set intersection.
$\subset$, $\subseteq$	Proper subset, Subset.
$\times$, S^d	Cartesian product, Cartesian product of d sets of S.
$\mathcal{P}$	Powerset.
$\mathbb{R}$,	The set of real numbers
$\mathbb{N}_{p:q}$	Natural numbers between the natural numbers p and q inclusive.
$\mathcal{P}_n(\mathbb{R})$	Vector space of polynomials of maximum order n over the real field.
$\odot$	Hadamard product.
$\frac{\partial}{\partial x}$, ∇_x	Partial derivative w.r.t. scalar x, Gradient w.r.t. tensor x.
M^T, $M_{i.}$, $M_{.i}$	Transpose of matrix M, i'th row of matrix M, i'th column of matrix M
$\overline{z}$	Complex conjugate of z.
$*$	Convolution.

Audio

ρ	Bit depth.
f_s, T_s, N_s	Sample rate, Duration of sample, Number of samples.
$f_{\mathcal{N}}$	Nyquist frequency.
s, s_n, $s^{(T)}$	Signal, Signal sample, Censored signal
F	Fourier transform.
$\tilde{s}$, $\hat{\tilde{s}}$	Frequency representation, Approximated frequency. representation

Probability Theory and Bayesian Inference

Ω, Σ, $\mathbb{P}$, $(\Omega, \Sigma, \mathbb{P})$	Sample space, Event space, Probability measure, Probability space.
$\mathbb{E}$, $\mathbb{E}_{x\sim p}$	Expectation, Expectation for x distributed by distribution p.
$\mathcal{K}$	Background knowledge.
$\mathbb{P}(h\|\mathcal{K})$, $p(\theta)$	Prior.
$\mathbb{P}(h\|\mathcal{D},\mathcal{K})$, $p(\theta\|\mathcal{D})$	Posterior.
$\mathbb{P}(\mathcal{D}\|h,\mathcal{K})$, $p(\mathcal{D}\|\theta)$	Likelihood.
$\mathbb{P}(\mathcal{D}\|\mathcal{K})$, $p(\mathcal{D})$	Evidence / Marginal likelihood.
$\mathcal{F}$	Variational free energy objective function.

Statistical Learning Theory

x, $\mathcal{X}$	Input, Input space.
y, $\mathcal{Y}$	Output, Output space.
$P(X,Y)$, $P(X)$, $P(Y\|X)$	Data generation process, Input distribution, Target distribution.
$\mathcal{M}$	Machine learning model.
f	Target function.
h, $\mathcal{H}$	Hypothesis, Hypothesis space.
θ, Ω_θ	Parameter, Parameter space.
ψ, Ω_ψ	Hyperparameter, Hyperparameter space.
$\mathcal{D}$, $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{val}}$, $\mathcal{D}_{\text{test}}$, $\tilde{\mathcal{D}}$	Dataset, Train dataset, validation dataset, Test dataset, Dataset space.
E_{train}, E_{val}, E_{test}	Training error , validation error, Testing error
$\mathcal{A}$	Learning algorithm.
L	Loss function
R, $\hat{R}$	Risk function, Empirical risk function.
g	Selected hypothesis.
λ, Λ	Regularisation Hyperparameter, Regulariser.
$\hat{J}$	Regularised empricial risk function.
J	Objective function

Deep Learning

t, $\mathcal{T}$	Learned transformation, Transformation space.
ℓ	Layer index.
n_ℓ	Number of nodes in layer ℓ.
σ, $\tilde{\sigma}$	Activation function, Activation function space.
w, W, $\mathcal{W}$	Weight vector, Weight matrix, Weight space.
b, $\mathcal{B}$	bias scalar/vector, Bias space.
a, c	Pre-activation, Post-activation
δ	Node Sensitivity
G	Stochastic gradient estimate
δ_d	Divisibility stability parameter
H, W, C	3D array's height, width and number of channels.
f_h, f_w	Kernel's height and width.
p_d, N_d	Dropout rate, Number of nodes using dropout.

1 Introduction

Artificial Neural networks, have found great success when applied to audio classification problems. The promise of these models is an ability to learn high level features from unstructured data, such as raw audio signal waveforms or their associated spectrograms. This mitigates the limitations and biases introduced by the human hand-designed features, such as Mel-Frequency Cepstrum Coefficents (MFCCs) and zero-crossing rate values, that have historically been used for these types of problems.
There are several characteristic attributes of audio data that make audio classification a particularly challenging machine learning problem. Most audio data's origin is continuous in nature, for example, recording audio via a microphone can be seen as a two stage process. First a continuous mechanical sound wave is generated at some physical source, which goes on to induce oscillations in the surrounding air pressure. These oscillations in air pressure propagate to the microphone and drive corresponding oscillations in the diaphragm of the microphone, attached to which is an electromagnet which then goes on to propagate the original sound waves oscillations as an electric current. The next stage is Analogue-to-Digital Conversion (ADC), this being the discretisation of the received continuous signal. Temporal discretisation is achieved by sampling the continuous signal at a rate called the sample rate f_s, while intensity discretisation is achieved by binning the signal into one of 2^ρ bins, this corresponding to the number of states representable by ρ bits, ρ is called the bit depth of the audio data.
In order for digital representations of audio data to be faithful representations of continuous source audio both ρ and f_s need to be as high as possible. The sample rate controls the resolution to which the frequency components of the signal can be represented. This can be mathematically quantified by the Nyquist frequency $f_N = \frac{f_s}{2}$ which represents the frequency above which frequency components' contributions to the signal get aliased onto lower frequency components. With this in mind most audio recording devices contain anti-aliasing filters which cut these higher frequency components out of the electrical signal before ADC. Thus, when presented with an audio signal of sample rate f_s, it is safe to assume that the highest frequency component recorded in this signal is f_N. The hearing range of humans is usually quoted as $\approx [20,\ 20 \times 10^3]$, this motivates one to use sample rates $f_s > 40kHz$. This is consistent with the most widely used sample rates, an example of which being $f_S = 44.1kHz$. This typical sample rate means that audio data is characteristically high dimensional, for example, a single minute $T = 60$ of data under this sample rate will result in a input vector of dimensionality in excess of 2.5×10^6. Pairing with a modest bit depth of $\rho = 16$ gives rise to a storage requirement of ≈ 5 MB per example. Thus, audio data is characteristically high dimensional and memory intensive to use.
The problem of audio data being characteristically high dimensional is exacerbated by the fact that labelling audio data is a fundamentally laborious task, this means most audio datasets are of a small to modest size. Paring the high dimensionality of audio data with the small to modest sized training data typically used in audio classification problems means that most models only explore a small fraction of the input space $\mathcal{X} \subseteq \mathbb{R}^d$. Forcing the model to make generalisations over the entire input space based on only a small number of training examples will inevitably lead to overfitting, i.e. the model learning training dataset specific patterns that do not generalise. Although, it should be noted that this is somewhat mitigated by the blessing of non-uniformity and the manifold hypothesis.
ANNs are highly flexible models with complex non-convex loss functions, an ANN typically trains by converging to some local minimum of this loss function. The smaller a training dataset is, the more numerous these high performing local minima are, due to their being

many combinations of the learnable parameters that led to high performing solutions, many of these solutions will overfit to the training data and not generalise to out-of-sample examples. A model that has many learnable parameter values that are highly consistent with the data is said to be under-specified by the data, choosing one of these learnable parameter values over another is rather arbitrary. Furthermore, if the hypothesis function, i.e. the mapping from inputs to outputs, associated with these parameters are functionally different, then it would be foolish to discard the potential improvement in performance that could be achieved by aggregating such hypotheses. The intuition behind the previous statement is that one hypothesis may perform well on one type of example that the other hypothesis performs poorly on, by aggregating the hypotheses one can mitigate each hypothesis' weaknesses.
Bayesian Neural Networks (BNNs) attempt to tackle this problem of under-specified models via Bayesian inference. Bayesian inference allows for the modelling of epistemic uncertainty, in this context this refers to the uncertainty associated with the learned parameters of the ANN. BNNs use the training data to update their prior beliefs about the distribution of learnable parameters to some posterior belief about the distribution of learnable parameters. BNNs can then exploit Bayesian Model Averaging (BMA) to aggregate the predictions associated with the different hypothesis functions resulting from probable learnable parameter values under the inferred posterior distribution of learnable parameters.
The above discussion about how audio classification problems often lead to under-specified ANNs and how BNNs are specifically suited to such problems, motivates the objective of this work. The main objective of this work is to investigate the differences in efficacy associated with comparable ANN and BNN architectures when applied to audio classification problems. The following work is split into a series of sections. The Literature Review section goes over relevant concepts from Bayesian inference, machine learning and deep learning required to adequately explain BNNs and their potential advantages. The Methodology section details the specific problem being tackled, alongside an extensive description of exactly how this problem tackled. The Results section shows a series of plots and tables summarising the main findings acquired during the execution of the method. The Discussion and Conclusion sections discuss and conclude the results presented in the previous section respectively. Furthermore, the code used in this project can be found on the GitHub repository: `https://github.com/jack-norrie/bayesian_deep_learning_for_audio_classification`.

2 Literature Review

2.1 Bayesian Inference

2.1.1 Bayesian Versus Frequentist Inference

Probability theory is branch of pure mathematics that can be formulated within a measure theoretic framework, this was first done by Kolmogorov in 1933. Kolmogorov's axioms for probability theory are equivalent to defining probabilities as measures $\mathbb{P}$ on a sigma-algebra Σ of outcomes Ω, which form a probability space $(\Omega, \Sigma, \mathbb{P})$, these being measure spaces with unit total measure $\mathbb{P}(\Omega) = 1$ [1, 2]. Statistical inference can be seen as an application of this branch of pure mathematics, the objective of which being to uncover statistical properties associated with phenomenon that one wishes to model. There are two main approaches to statistical inference, the Bayesian and the frequentist approach, the latter being the status quo and the former being considered controversial by some [3]. These two approaches differ fundamentally in what they are using probability theory to model and by extension how probabilities should be interpreted.

The frequentist approach uses probability theory to model limiting relative frequencies. The interpretation of such probabilities requires one to envision a theoretical thought experiment. Consider an experiment in which one observes the outcome of some probabilistic event E, whereby the event either occurs or does not occur. Now suppose one could repeat this experiment an infinite number of times under the exact same conditions, this is called the repeated sampling principle [4]. Then, the probability of an event is considered to be the long run proportion of times the event occurred compared to the total number of runs of the experiment [3]. This interpretation of probabilities puts some limitations on the types of problems one can apply statistical inference to. For example, suppose one is using a family of models parameterized by some parameter vector $\theta \in \Omega_\theta$, then under the assumption that at least one of the models in this family of models is an accurate description of reality, there exists some true $\tilde{\theta} \in \Omega_\theta$. Since this true $\tilde{\theta}$ does not vary between experimental runs under the repeated sampling principle, it does not make sense to discuss this true parameter value as a probabilistic quantity, the true parameter is fixed and unknown.

The Bayesian approach uses probability theory to model degrees of belief with respect to logical propositions [3]. This usage of probability theory is quite far removed from the conventional frequentist usage of probability theory, as such, it is reasonable to question its validity. Fortunately, in 1946 Cox [5] showed that this usage of probability theory is valid if one is to construct a system of assigning degrees of belief that conforms to a simple set of axioms encoding "common sense" [6, 7]. The probabilities arising within this framework can then be interpreted as the degree to which a Bayesian believes a given logical proposition to be true based on their prior background knowledge. If a logical proposition can have a meaningful probability assigned to it based on available background knowledge then it is said to be sufficiently conditioned. Typically, the set of logical propositions that are sufficiently conditioned based on a typical Bayesian's background knowledge allow for Bayesian inference to be applied to a larger variety of problems than that of frequentist inference. For example, it is now feasible to assume that a Bayesian could use their background knowledge to ascribe some meaningful value to the probability of the logical proposition:

$$h = \text{"The parameter vector } \theta = \theta^*\text{."}$$

Central to the Bayesian approach to statistical inference is Bayes theorem, this provides a means for Bayesian's to update their beliefs about some hypothesis h, based on new data $\mathcal{D}$ and their prior background knowledge $\mathcal{K}$. Bayes theorem can be seen in Equation 1 [8].

$$\mathbb{P}(h|\mathcal{D},\ \mathcal{K}) = \frac{\mathbb{P}(\mathcal{D}|h,\ \mathcal{K})\mathbb{P}(h|\mathcal{K})}{\mathbb{P}(\mathcal{D}|\mathcal{K})} \tag{1}$$

The key components of Equation 1 are:

- Prior: $\mathbb{P}(h|\mathcal{K})$
- Posterior: $\mathbb{P}(h|\mathcal{D},\ \mathcal{K})$
- Likelihood: $\mathbb{P}(\mathcal{D}|h,\ \mathcal{K})$
- Evidence: $\mathbb{P}(\mathcal{D}|\mathcal{K})$

With this nomenclature in place one can view Bayes theorem within the context of Bayesian inference as an update rule, transforming a prior into a posterior distribution. This update rule can be used iteratively as new data becomes available, the old data simply becomes part of the conditioning background knowledge and the update rule can be reapplied with the new data taking the place of the old data.

2.1.2 Prior Selection

The main controversy associated with Bayesian inference is the seeming subjectivity associated with the choice of prior. Bayesian's argue that the framework of Bayesian inference is self consistent, stating that differences in choice of prior can only arise due to differences in conditioning background knowledge $\mathcal{K}$ [7]. They argue that two Bayesian's with identical conditioning information should come to the same conclusions. Objective Bayesians develop these ideas further by outlining principles for selecting priors based on ones available background knowledge. An example of such a principle is the principle of maximum entropy, which states the prior should be selected such that its entropy is maximal over the class of priors consistent with ones background knowledge, where the entropy of a distribution p is defined by Equation 2 [8, 9].

$$H(p) \equiv -\mathbb{E}_{\theta\sim p}\left(\log(p(\theta))\right) \tag{2}$$

Although the objective Bayesian approach is very appealing there are many circumstances where a prior is selected purely for computational convenience. A key example of such priors are conjugate priors whereby the form of the posterior is the same as the prior, updating the prior then becomes a case of evaluating some algebraic expression for the new parameters of the posterior distribution. This approach of choosing a computationally convenient prior is the most relevant to the current literature on Bayesian Neural Networks. As such, the subjectivity criticism is somewhat valid in this circumstance. Therefore, for conciseness the formality of explicitly including the conditioning background information for the terms in Equation 1 will be omitted from this point forward.

2.1.3 Bayesian Variational Methods

Bayes theorem is a deceptively simple equation, in practice one often does not have access to the form of the evidence term of this equation. Suppose the hypothesis space $\mathcal{H}$ of hypothesis h under consideration is isomorphic to some parameter space Ω_θ such that there is a bijective relationship between each $\theta \in \Omega_\theta$ and each $h \in \mathcal{H}$. Then, one can use the law of total probability to compute the evidence term. It is for this reason that another common term for the evidence term is the marginal likelihood. Equation 3 shows the form of the marginal likelihood [3].

$$\mathbb{P}(\mathcal{D}) = \int_{\Omega_\theta} \mathbb{P}(\mathcal{D}|\theta)P(\theta)d\theta \tag{3}$$

Typically the integral displayed in Equation 3 does not have an analytic solution and must be approximated via numerical methods. Standard quadrature methods have convergences rates that scale inverse exponentially as the dimensionality of this integral increases, this coinciding with the exponential increase in spacing between points in an equispaced sample grid as the dimensionality of its space increased. This makes calculation of the marginal likelihood via quadrature methods computationally intractable for high dimensional parameter spaces [10].

High dimensional parameter spaces are commonplace in Bayesian Inference. As such, efficient methods for calculation/approximation of posterior distributions are paramount to this field. There are two prominent families of methods commonly used for these problems, Markov-Chain Monte-Carlo (MCMC) and Variational methods. The former family of methods generate ergodic Markov chains with stationary distributions equal to the target posterior distribution, samples generated from this Markov chain can then be used to generate Monte-Carlo estimates of statistics associated with the target posterior distribution. The latter family of methods convert the problem of approximating the posterior into an optimisation problem. Variational methods tend to be faster and scale to larger datasets better than MCMC methods, although they do not have the same asymptotic guarantees associated with MCMC methods [11]. Training Bayesian neural networks is an iterative processes whereby the posterior distribution must be estimated many times, often whilst using large datasets, it is for this reason that variational methods are the most relevant for these models.

Variational methods involve approximating some target distribution p by some variational distribution q, where q belongs to a chosen family of variational distributions Q. The selected variational distribution $q^* \in Q$ minimises the Kullback–Leibler (KL) Divergence between itself and the posterior distribution, this is summarised in Equations 4 and 5 [11, 12].

$$q^* = \underset{q \in Q}{\operatorname{argmin}} D_{KL}(q\|p) \tag{4}$$

$$D_{KL}(q\|p) \equiv \mathbb{E}_{x\sim q}\left(\log\left(\frac{q(x)}{p(x)}\right)\right) \tag{5}$$

The family of variational distributions is typically chosen such that calculation of the KL divergence is computationally tractable. A common simplifying assumption for high dimensional distributions is the so called mean-field approximation, this assumes the distribution of q factorises. This assumption is summarised in equation 6

$$q(x) = \prod_i q_i(x_i), \ \forall q \in Q \tag{6}$$

The KL divergence is often interpreted as a measure of distance between two probability distributions. Indeed, the KL divergence conforms two of the three requirements of a metric, these being positive definiteness and the triangle inequality [13]. However, inspection of Equation 5 immediately reveals that the KL divergence is not symmetric for all p and q. Consequently, the KL divergence is technically only a qausi-metric rather than a metric on probability distributions [12]. The consequences of this asymmetry when optimising over $q \in Q$ are summarised in Table 1.

	$D_{KL}(p\|q)$	$D_{KL}(q\|p)$
$p \gg q$	High Penalty	Negligibly Preferable
$p \approx q$	Approximately Zero Penalty	Approximately Zero Penalty
$p \ll q$	Negligibly Preferable	High Penalty

Table 1: Penalties associated with different regions of the target distribution and variational distributions relative values.

This asymmetry means it is important to select the correct order of arguments in the KL divergence for the problem one is trying to tackle, the two variations being the forward KL divergence $D_{KL}(p\|q)$ and the reverse KL divergence $D_{KL}(q\|p)$. Minimising the forward KL divergence will favour a variational distribution that assigns high probability to regions where the target distribution is high. Furthermore if $p > 0$ over some region then q being zero will cause an infinite KL divergence, the forward KL divergence is said to be zero avoiding. This zero avoiding behaviour causes the variational distribution to "cover" the target distribution and hence over-estimate its variance. Conversely, minimising $D_{KL}(q\|p)$ will favour a variational distribution that avoids placing high probability to regions where the target distribution is low. Similarly, if p is zero over a region then q is forced to be zero to avoid an infinite KL divergence, the reverse KL divergence is said to be zero forcing. This zero forcing behaviour causes the variational distribution to hone onto a specific mode of the target distribution and hence under-estimate its variance [11, 14]. These behaviours are demonstrated in Figure 1.

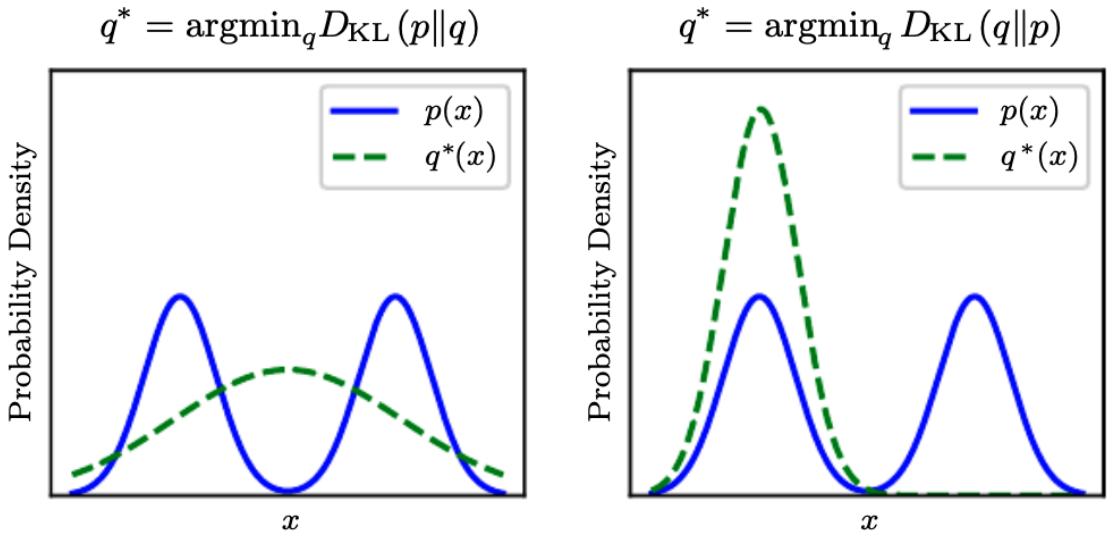

Figure 1: [12] Minimisation of the forward and reverse KL divergences for a variational family of univariate Gaussians and a Gaussian mixture target.

Minimisation of the reverse KL divergence is the most relevant for Bayesian neural networks. This is because one can factor out the computationally intractable marginal likelihood as follows:

$$\begin{aligned}
D_{KL}(q(\theta)\|p(\theta|\mathcal{D})) &= \mathbb{E}_{\theta\sim q(\theta)}\left(\log\left(\frac{q(\theta)}{p(\theta|\mathcal{D})}\right)\right) = \mathbb{E}_{\theta\sim q(\theta)}\left(\log\left(q(\theta)\right) - \log\left(p(\theta|\mathcal{D})\right)\right) \\
&= \mathbb{E}_{\theta\sim q(\theta)}\left(\log\left(q(\theta)\right) - \log\left(\frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}\right)\right) \\
&= \mathbb{E}_{\theta\sim q(\theta)}\left(\log\left(q(\theta)\right) - \log\left(p(\mathcal{D}|\theta)\right) - \log\left(p(\theta)\right) + \log\left(p(\mathcal{D})\right)\right) \\
&= \mathbb{E}_{\theta\sim q(\theta)}\left(\log\left(\frac{q(\theta)}{p(\theta)}\right)\right) - \mathbb{E}_{\theta\sim q(\theta)}\left(\log\left(p(\mathcal{D}|\theta)\right)\right) + \log\left(p(\mathcal{D})\right) \\
&= D_{KL}(q(\theta)\|p(\theta)) - \mathbb{E}_{\theta\sim q(\theta)}\left(\log\left(p(\mathcal{D}|\theta)\right)\right) + \log\left(p(\mathcal{D})\right)
\end{aligned}$$

Since the log marginal likelihood is an additive constant with respect to optimisation over $q \in Q$, it can be ignored in the optimisation procedure, this is summarised as follows:

$$\underset{q\in Q}{\operatorname{argmin}}\left(D_{KL}(q(\theta)\|p(\theta|\mathcal{D}))\right) = \underset{q\in Q}{\operatorname{argmin}}\Big(D_{KL}(q(\theta)\|p(\theta)) - \mathbb{E}_{\theta\sim q}\left(\log\left(p(\mathcal{D}|\theta)\right)\right)\Big).$$

This leads to the variational free energy objective function, which is defined in equation 7 [15].

$$\mathcal{F}(q;\mathcal{D}) \equiv \mathbb{E}_{\theta\sim q}\left(-\log\left(p(\mathcal{D}|\theta)\right)\right) + D_{KL}(q(\theta)\|p(\theta)) \tag{7}$$

Importantly, the variational free energy loss function can be evaluated without calculation of the computationally intractable marginal likelihood $p(\mathcal{D})$. Additionally the loss function can be seen to be composed of two components:

- Likelihood Penalty: $\mathbb{E}_{\theta\sim q}\left(-\log\left(p(\mathcal{D}|\theta)\right)\right)$
- Complexity Penalty: $D_{KL}(q(\theta)\|p(\theta))$

The likelihood penalty is a data dependent term, it gives the selected q^* a propensity to minimise the expected negative-log-likelihood of the data, or equivalently maximise the expected log-likelihood. The complexity penalty favours against variational distributions which deviate too heavily from the prior. A consequence of the combination of these two terms is that the selected variational distribution q^* can only vary significantly from the prior if there is sufficient evidence from the data to justify this deviation, this evidence manifesting as a variational distribution which is highly consistent with the data and thus promoting a high expected log-likelihood [15].

2.2 Machine Learning

2.2.1 Overview

There is no single widely accepted definition of machine learning. Furthermore, as this contemporary field has evolved over the years so to have the widely accepted definitions. An example of a modern and well accepted definition of what the field of machine learning encompasses is given below:

> "*we define machine learning as a set of methods that can automatically detect patterns in data, and then to use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty*"
> - Kevin Murphy, 2012 [14].

This is quite a broad definition, and as such machine learning can be applied to a plethora of problems. These problems typically fall into one of the following three learning paradigms:

- Unsupervised Learning - model trains on a set of inputs, with the objective of learning some structure or pattern associated with the inputs [16].
- Supervised Learning - model trains on a set of input-output pairs, with the objective of making output predictions for unseen inputs. It is called supervised learning because generation of the dataset requires a "supervisor" to label the outputs associated with the training inputs, for this reason outputs are often referred to as labels [16].
- Reinforcement Learning - model learns how to control an agent that can interact with their environment, this can typically be modelled as a Markov Decision Process (MDP). For any given state of the environment the agent has a set of actions they can take. The agent will receive a reward signal based on the action it takes for a given environmental state. After an action is taken the environment moves to a new state. The objective of the model is to learn a policy, detailing which actions an agent should take in a given state, such that the agent maximises its expected long-term reward [17].

2.2.2 Supervised Learning

Supervised learning problems can usually be placed within a common framework. As previously mentioned, supervised learning deals with problems for which the data set $\mathcal{D}$ consists of a set of input-output pairs $\mathcal{D} = \{(x_i, y_i) : 1 \leq i \leq N\}$. The inputs and outputs are elements of an input and output space $\mathcal{X}$ and $\mathcal{Y}$ respectively. The dataset is thus a member of the dataset space $\tilde{\mathcal{D}} \equiv \{\mathcal{D} \in \mathcal{P}(\mathcal{X} \times \mathcal{Y}) : |\mathcal{D}| < \infty\}$. The problem is said to be a classification problem if $\mathcal{Y}$ is discrete and a regression problem if $\mathcal{Y}$ is continuous [16].
It is assumed there exists some theoretical target function $f : \mathcal{X} \to \mathcal{Y}$ that maps inputs to outputs. More generally, this target function can be noisy, meaning it does not give a deterministic output for a given input, this gives rise to a target distribution $P(Y|X)$. With this idea of a noisy target function in mind, the data can be thought to be generated from independent and identically distributed samples (x_i, y_i) from some data generation process $P(X, Y) = P(Y|X)P(X)$, where $P(X)$ denotes the input distribution [16].
The dataset is then fed into a machine learning model. A machine learning model $\mathcal{M}$ is characterised by the following 3 components:

- Representation - The formal mathematical description of how the model assigns outputs to inputs, this can be represented via some hypothesis function $h : \mathcal{X} \to \mathcal{Y}$. Typically, the representation encompasses many hypothesis functions, these form the hypothesis set $\mathcal{H}$. Parametric representations have hypothesis sets who are isomorphic to some M dimensional parameter space $\mathcal{H} \simeq \Omega_\theta \subseteq \mathbb{R}^M$ [16, 18].
- Evaluation - The means by which hypothesis functions are compared and ranked. A common approach is to utilise a point wise loss function $L : \mathcal{H} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ or $L_h : \mathcal{Y}^2 \to \mathbb{R}$, whereby a hypothesis which produces a lower loss is optimal. The evaluation method will make implicit assumptions about how the model perceives a hypothesis' "closeness" to the optimal target function [16, 18].
- Optimisation - The method used to search through the hypothesis set and select an optimal hypothesis $g \in \mathcal{H}$, this is also referred to as the learning algorithm $\mathcal{A}$ [16, 18].

The way that these components of a typical supervised learning problem interact are summarised in Figure 2.

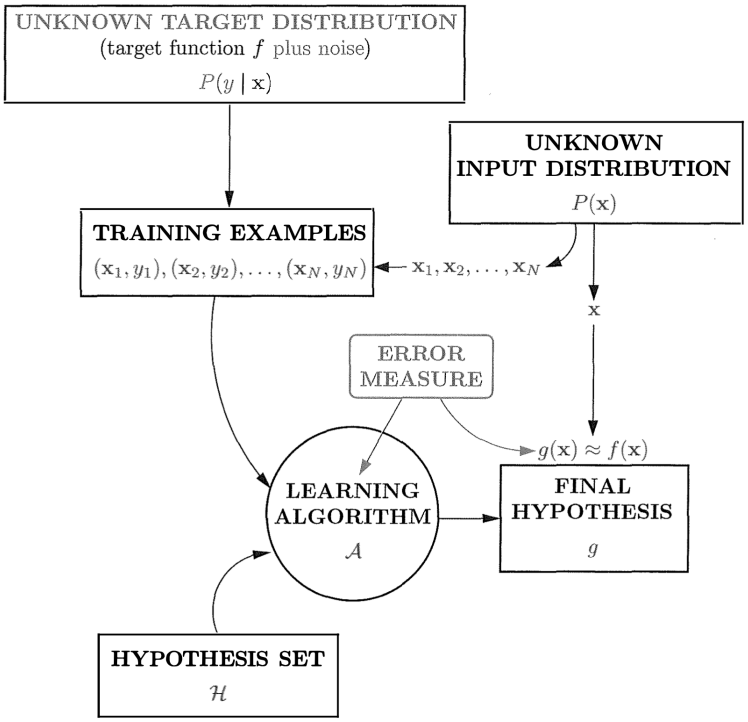

Figure 2: [16] Diagram displaying how the different components of a typical supervised learning problem interact.

It is very important to select an appropriate machine learning model for a given problem. The chosen representation needs to be complex enough to describe the phenomenon associated with the problem, but not too complex such that overfitting becomes a significant problem. The evaluation method needs to reflect the goals that one is trying to accomplish, for example, choice of loss function gives rise to propensities for the learning algorithm's selected hypothesis. The optimisation method needs to be appropriate for the time, memory and data constraints of the problem. If all three of these components are carefully considered then the objectives of supervised learning can be fulfilled, mainly, that the selected hypothesis approximates the target function/distribution [16, 18].

2.2.3 Generalisation

The objective of supervised learning is to use training data $\mathcal{D}$ to learn generalisations about the underlying data generation process $P(X,\ Y)$. Typically, this corresponds to using the training data to select some hypothesis g that approximates the behaviour of the target function/distribution, this approximation can then be used to infer the labels of unseen inputs. The sense in which g approximates the target distribution is implicitly encoded by the choice of loss function.
The aforementioned objective can then be restated as a desire for the performance of a machine learning model, as determined by the loss function, to generalise from seen in-sample data $\mathcal{D}$ to unseen out-of-sample data. To discuss out-of-sample behaviour further consider the random vector $(X, Y) \sim P(X,\ Y)$, then, the out-of-sample performance for a hypothesis h can be considered to be a random variable $\mathcal{L}_h = L(h(X), Y) \equiv L(h(X), Y)$. A natural location summary statistic for the distribution of $\mathcal{L}_h$ would be its expectation value, this is called the risk function, it is defined in Equation 8 [16, 19].

$$R(h) = \mathbb{E}_{(X,Y)\sim P(X,Y)}(L(h(X), Y))) \tag{8}$$

Recalling that the dataset $\mathcal{D}$ is assumed to be composed of N independently and identically distributed samples $(x_i,\ y_i) \overset{\text{i.i.d}}{\sim} P(X,\ Y)$. Then, assuming the expectation in Equation 8 is finite, it follows from the strong law of large numbers that the Monte-Carlo estimate shown in Equation 9 converges almost surely to the risk function as $N \to \infty$ [2]. Furthermore, if the second moment of the loss random variable is assumed finite then the central limit theorem dictates that this estimate will converge at a rate $O\left(\frac{1}{|\mathcal{D}|}\right)$ [10]. This Monte-Carlo estimate is referred to as the empirical risk function [16, 19].

$$\hat{R}(h;\ \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}}^{N} L(x, y, h) = \frac{1}{N} \sum_{i=1}^{N} L(x_i, y_i, h) \tag{9}$$

Most machine learning models are attempting to perform empirical risk minimisation, whether or not they converge to a local minimum or global minimum in this pursuit will be determined by the form of the loss function and the learning algorithm used. Empirical risk minimisation is summarised in Equation 10 [16, 19].

$$g = \underset{h\in\mathcal{H}}{\operatorname{argmin}}\ \hat{R}(h;\ \mathcal{D}) \tag{10}$$

Unfortunately, once one starts optimising the empirical risk over the hypothesis set, then the resulting estimator is biased to under-estimate the risk of the selected hypothesis. This can be summarised as follows:

$$\mathbb{E}\left(\hat{R}(g;\ \mathcal{D})\right) = \mathbb{E}\left(\min_{h\in\mathcal{H}} \hat{R}(h;\ \mathcal{D})\right) < R\left(\underset{h\in\mathcal{H}}{\operatorname{argmin}}\ \hat{R}(h;\ \mathcal{D})\right) = R(g),\ \text{for } N < \infty$$

Intuitively, this can be understood by the possibility of a specific hypothesis performing very well on a specific dataset $\mathcal{D}$, however this performance may not be representative of this hypothesis' performance on the out-of-sample data. Empirical risk minimisation has a propensity to select such hypotheses, especially in this hypothetical case were the chosen model performs disproportionately well on the specific dataset available.
The above is problematic if one not only wishes to train their model, but to also generate an accompanying unbiased out-of-sample performance estimate. A common strategy to

estimate the out-of-sample performance of a model is to randomly split the dataset into a training and testing dataset such that $\mathcal{D} = \mathcal{D}_{\text{train}} \bigcup \mathcal{D}_{\text{test}}$. Next the training dataset is fed into the learning algorithm, this processes is called training the machine learning model. Once training is complete and the learning algorithm has selected a hypothesis g from the hypothesis set, the empirical risk of g on the testing dataset is evaluated. This gives rise to the train-set and test-set errors shown in Equations 11 and 12 respectively [16].

$$E_{\text{train}} = \hat{R}(g;\ \mathcal{D}_{\text{train}}) \tag{11}$$

$$E_{\text{test}} = \hat{R}(g;\ \mathcal{D}_{\text{test}}) \tag{12}$$

2.2.4 Overfitting

A closely related concept to generalisation is the so called generalisation gap, this being the difference between the test-set error and the train-set error. The generalisation gap typically grows as the complexity of a model's hypothesis set increases. Overfitting is the term used to describe this phenomenon of a large generalisation gap, this referring to the scenario where the model has overfit to the available data and has not been able to generalise the patterns it has learnt to out-of-sample data. Some of the most important results in machine learning place probabilistic concentration bounds on the generalisation gap associated with a model, an example of such a result is the famous Vapnik-Chervonenkis inequality for binary classification problems.
This phenomenon of overfitting can be understood as follows. As the complexity of a model's hypothesis set grows, so too does a model's ability to find fictitious patterns that work well on a specific dataset, but which do not generalise to out-of-sample data. An example of a fictitious pattern would be a model fitting to the noise present in the samples generated by a noisy target. It is a common misconception that all such fictitious patterns are a result of fitting to noise, however this is not necessarily the case. A complex hypothesis set will often encompass such a diverse set of hypotheses that there is likely to be at least one hypothesis in $\mathcal{H}$ that can perform disproportionately well on a specific dataset $\mathcal{D}$ of a specific size N. For example, if the hypothesis set is the set of polynomials of highest degree equal to $(N-1)$ then $\mathcal{H} = \mathcal{P}_{N-1}(\mathbb{R})$, then this hypothesis set will be able to perfectly fit any training dataset of size N [20], it is very unlikely that this perfect performance will generalise to out-of-sample data. Figure 3 demonstrates this behaviour.

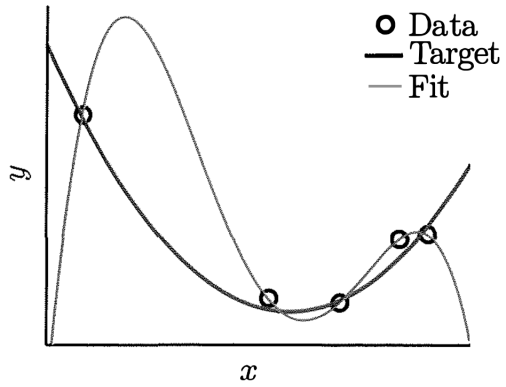

Figure 3: [16] The 5'th order polynomial perfectly fits training data, but does not resemble the target function, this results in poor out-of-sample performance.

A natural way to reduce a models capacity to produce these specialised hypotheses is to increase the size of the dataset. Any potential fictitious patterns must now be consistent with a larger number of data points, thus decreasing the efficacy of hypotheses which focus on such patterns. Furthermore, increasing the signal to noise ratio of the data will improve the overfitting problem, this will help promote hypotheses which focus on fitting patterns associated with the signal in the dataset, such patterns should generalise to out-of-sample data, unlike fictitious patterns associated with the noise present in the training data. Unfortunately, the size and signal-to-noise ratio of the dataset are not something a machine learning practitioner is typically in control of [16].

2.2.5 Regularisation

Although a machine learning practitioner has limited control over the size and quality of their data, they do have control over the components of the machine learning model they use, more specifically the learning algorithm associated with their selected machine learning model. Regularisation is an overarching term used to describe modifications one can make to the machine learning algorithm's learning algorithm in order to mitigate the issue of overfitting [12].
A common form of regularisation is to change the learning algorithms goal from empirical risk minimisation to structural risk minimisation. This involves modifying the risk function to include some complexity penalty term, giving rise to a regularised empirical risk function $\hat{f}$. This complexity term generally takes the form $\lambda\Lambda(h)$, where $\lambda \in \mathbb{R}^+$ and $\Lambda : \mathcal{H} \to \mathbb{R}^+$. The complexity function Λ should be selected such that high capacity subsets of the hypothesis set are more heavily penalised than low capacity subsets. The effect of the complexity term is then to increase the propensity of the learning algorithm to select simple models that are consistent with the data. The objective of structural risk minimisation, in terms of selection of a hypothesis, is displayed in Equation 13 [16].

$$g = \underset{h\in\mathcal{H}}{\operatorname{argmin}}\left(\hat{f}(h;\mathcal{D})\right) = \underset{h\in\mathcal{H}}{\operatorname{argmin}}\ \left(\hat{R}(h;\ \mathcal{D}) + \lambda\Lambda(h)\right) \tag{13}$$

There are two very prominent complexity penalties relevant to parametric machine learning model representations, these being the L1 and L2 complexity penalties:

$$\text{L1 Regularisation: } \Lambda(\theta) = \|\theta\|_1,$$
$$\text{L2 Regularisation: } \Lambda(\theta) = \|\theta\|_2^2.$$

The associated structural risk minimisation problem, under sufficient regularity conditions for the empirical risk function, corresponds to a constrained optimisation problems via Lagrange multipliers. L1 and L2 regularisation correspond to constraining the parameter vector to a hypercube and hypersphere respectively. The volume of these constraining manifolds is determined by the regularisation hyperparameter λ. The higher λ is the greater the complexity penalty, and hence the smaller the smaller the volume of the constraining manifold [16].

2.2.6 Data Augmentation

As was stated previously, a machine learning practitioner is not in control of the size of their dataset. Gathering such a dataset, especially in a supervised learning setting where a supervisor is often required to manually label each example in the dataset, can be a

laborious and expensive process. As such, it is usually not feasible to generate more data from the data generation process $P(X, Y)$ [21].
One way that machine learning practitioners get around this is through the process of data augmentation. In essence they build a model of the data generation process $\hat{P}(X, Y)$ using the available dataset $\mathcal{D}$ and any background knowledge they posses on the data generation process $\mathcal{K}$. This background knowledge can be used to reduce the machine learning model's propensity to find specific fictitious patterns. For example, if one was training on images of flowers, the data augmentation process could involve flipping or rotating the images slightly. This would penalise hypotheses using these features to overfit to specific datasets. This sort of data augmentation is demonstrated in Figure 4 [21].

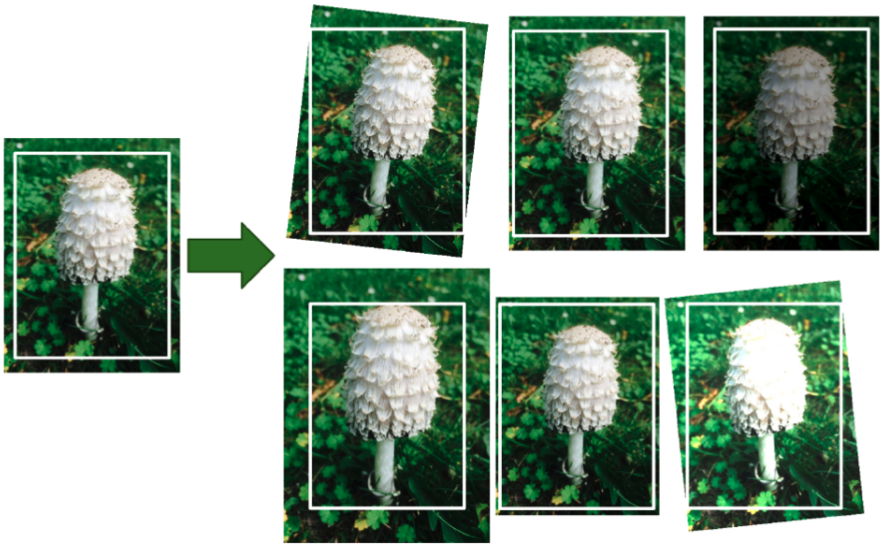

Figure 4: [21] Example of a data augmentation processes. The diagram displays 6 samples produced by the model of the data generation process and the original image that these samples are based on. Several augmentations are displayed, these including rotations, translation, magnification and changes in brightness.

Once a model of the data generation process has been built it can be used to simulate as many examples as desired. Obviously, samples generated from the true data generation process are preferable to the samples generated from this model of the data generation process. After all, if the machine learning practitioner had a sufficient understanding of the data generation process such that this was not the case, then there would be no need for them to be using machine learning in the first place. Furthermore, the model of the data generation processes must be accurate for this to produce any benefit. It is possible for data augmentation to produce an adverse effect if the assumed background knowledge encoded into the model of the data generation process is incorrect. Ideally, the generated samples should be indistinguishable in origin from those generated by the true data generation process [21].
Data augmentation can be viewed as a modification to the learning algorithm. One could imagine data augmentation being performed by the learning algorithm shortly after the non-augmented data is fed into it. In fact, this is exactly how some machine learning practitioners implement data augmentation, whereby the modelled data generation process is used to effectively generate a new dataset at each iteration of training. For this reason, data augmentation is actually considered a form of regularisation [21].

2.2.7 Validation and Cross-Validation

Sometimes, especially in scenarios of limited data, holding out data to form a test-set can be impractical. As has been discussed previously overfitting can be mitigated by increasing the size of the dataset used for training. A model trained on more data will generally out-perform one trained on less data, due to the associated reduction in generalisation gap. From a pure performance perspective a machine learning practitioner could be convinced to use the entire dataset for training, believing that the compromise to performance of generating an accurate out-of-sample performance estimate by holding out a test-set would be too great.
Fortunately, one does not need to give up on generating an accurate out-of-sample error estimate when training on the entire dataset. Validation outlines a procedure to produce such an estimate while also training the final model on the full dataset [16]. First , the dataset is partitioned into a training dataset and a validation set $\mathcal{D} = \mathcal{D}_{\text{train}} \bigcup \mathcal{D}_{\text{val}}$. Next the learning algorithm is fed the training dataset and selects a hypothesis $g^{(-)}$. This hypothesis is then evaluated on the validation dataset, this produces the validation error, the equation for which is displayed in Equation 14.

$$E_{\text{val}} = \hat{R}(g^{(-)};\ \mathcal{D}_{\text{val}}) \tag{14}$$

After this out-of-sample error estimate is produced the model is trained on the full dataset $\mathcal{D}$ to produce a selected hypothesis g. Unlike the training error, the validation error actually has a pessimistic bias, since the final model trained on the full dataset will likely out-perform the model trained on the training dataset, this being the model whose performance the validation error forms an unbiased estimate for [16]. There are essentially two stages of approximation in relating the validation error to the out-of-sample performance of the model trained on the full dataset:

$$E_{\text{val}} \approx R(g^{(-)}) \approx R(g) \text{ [16].}$$

The first approximation is aided by increasing the size of the validation dataset, this reduces the variance of the empirical risk estimator associated with $g^{(-)}$. Conversely the second approximation is aided by increasing the size of test set such that the training conditions for selecting $g^{(-)}$ are more comparable to those for selecting g, i.e. this reduces the pessimistic bias. This can be understood as a bias-variance trade off, controlled by the size of the validation dataset [16].
K-fold Cross-validation provides a means for reaping the benefits of a low bias estimate associated with a large training dataset, while simultaneously mitigating the associated increase in variance that comes along with reducing the size of the validation set [16]. The dataset is partitioned into K equally sized sub-datasets called folds such that $\mathcal{D} = \bigcup_{i=1}^{K} \mathcal{D}^{(i)}$. Associated with each fold is a training fold $\mathcal{D}^{(-i)} = \mathcal{D} \setminus \mathcal{D}^{(i)}$ equal to the full dataset with a specific fold excluded. K models are trained on each of the K training folds to produce selected hypotheses $g^{(i)}$. These selected hypotheses are then tested on their associated fold that was exempt from training. The mean of these fold-wise validation errors is taken to produce the K-fold cross validation error estimator, its form is given by equation 15 [16].

$$E_{\text{cv}} = \frac{1}{K} \sum_{i=1}^{K} E_{\text{val}}^{(i)} = \frac{1}{K} \sum_{i=1}^{K} \hat{R}(g^{(i)};\ \mathcal{D}^{(i)}) \tag{15}$$

Each of the fold wise validation accuracies can have the same arguments applied to them as the previous validation accuracies. A large training fold size will result in less bias relative

to $R(g)$, however the variance of the estimator will increase correspondingly as the fold upon which testing is conducted decreases in size [22]. However, the increase in the fold-wise validation accuracies' variances is now mitigated by the reduction in variance caused by aggregating these fold-wise validation errors when forming the cross validation error. This mitigation typically cannot counteract the increasing fold-wise validation accuracies' increase in variance indefinitely, as such there still exists a bias-variance trade-off, this time controlled by K. Additionally, there is the subtlety that the fold-wise validation errors are correlated, since the training-folds overlap, i.e. $\mathcal{D}^{(-i)} \bigcap \mathcal{D}^{(-j)} \neq \{\ \}$, this causing an increase in the estimators variance. Empirically, values of $K = 5$ and $K = 10$ have been found to work well in most situations, producing better out-of-sample error estimates than simple validation error estimates [22]. The only downside is that using K-fold cross-validation is approximately K times more computationally expensive than using validation [6].

2.2.8 Hyperparameter Tuning

Hyperparameters $\psi \in \Omega_\psi$ are parameters that are decided upon before learning commences, they can be seen as design choices for the machine learning model $\mathcal{M}(\psi)$. Unlike learnable parameters $\theta \in \Omega_\theta$, they do not change throughout the course of training. The previous section on regularisation introduced the regularisation hyperparameter λ, it was seen that this played an important role in constraining the hypothesis set. With this importance in mind, it would surely be advantageous to formulate a systematic method for selecting / tuning these hyperparameters.
A naive first attempt at this problem would be to perform some kind of search over a finite subset of the hyperparameter space $\Psi \subseteq \Omega_\psi$, selecting the hyperparameter that minimises the test-set error. However, as was the case when optimising over the training dataset, after optimising over the testing dataset the associated test-set error becomes an under-estimate for the true out-of-sample performance. As a general rule of thumb, the test set should be completely untouched until the final evaluation of the model.
Ideally, a method of tuning hyperparameters that only uses the training dataset $\mathcal{D}_{\text{train}}$, while simultaneously using the entire training dataset to fit the learnable parameters of the model, is desired. This is exactly the type of problem that validation / cross-validation was designed for. One can either partition the training dataset into a sub-training dataset and validation dataset $\mathcal{D}_{train} = \mathcal{D}_{\text{sub-train}} \bigcup \mathcal{D}_{\text{val}}$ to generate a validation error estimate for the out-of-sample error for each $\psi \in \Psi$, or alternatively one can partition the training dataset into K folds such that $\mathcal{D}_{train} = \bigcup_{i=1}^{K} \mathcal{D}^{(i)}$ to generate K-fold cross validation error estimates for the out-of sample error for each $\psi \in \Psi$. Minimisation over these validation / cross-validation out-of-sample error estimates can then be performed with respect to the hyperparameters. The model is then retrained on the entire training dataset using the selected hyperparameter. At no point during training has the learning algorithm accessed the training set, as such, the test-set error of this final selected hypothesis is an unbiased estimator for the out-of-sample error.
The above procedure effectively turns the hyperparameters into learnable parameters, whose optimisation is conducted by minimisation of validation / cross-validation error estimates generated from the training data. As such, it is possible to overfit with respect to the hyperparameter tuning processes. This problem increases in severity as the number of hyperparameters increases, however this problem is usually negligible when optimising over only a few hyperparameters.

2.3 Deep Learning

2.3.1 Overview

Deep learning is a sub-field of machine learning, more specifically it is a type of representation learning. Classical machine learning models typically involve feeding a learning algorithm a set of hand-designed features, these features z are generated by performing some transformation $t : \mathcal{X} \rightarrow \mathcal{Z}$ to the inputs x of the dataset in order to generate a more insightful representation of the data. These hand-designed features are usually constructed by invoking the machine learning practitioner's background knowledge. The objective is to design features that correspond to factors of variation in the observed dataset [12, 23], such features are very informative and can naturally cluster the dataset into a representation that lends itself to pattern discovery by a machine learning model. Rather than hand-designing these features, representation learning models seek to learn these transformations [12]. Deep learning models take this a step further by composing a sequence of learned transformations. This produces a hierarchical representation of the data, each learned transformation extracting a more abstract set of features from the data than the last [12]. Figure 5 demonstrates the differences in these types of models.

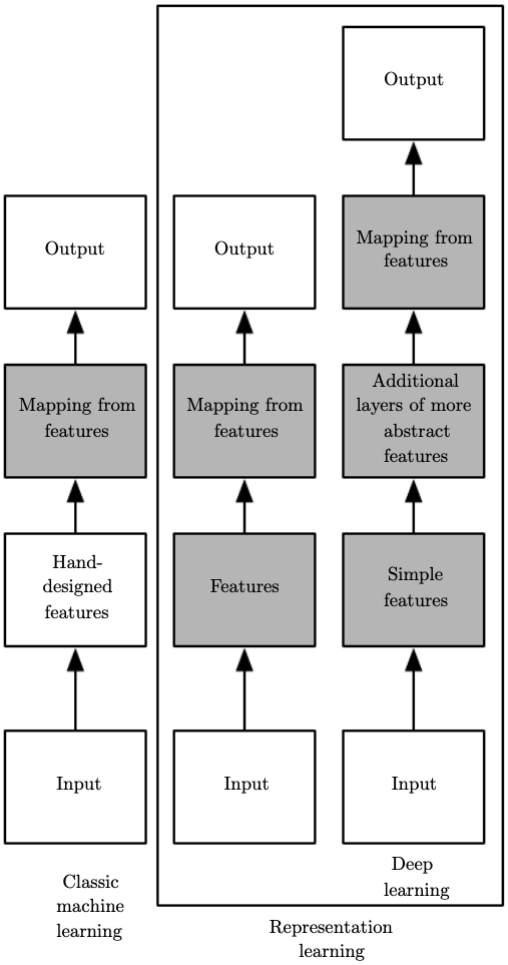

Figure 5: [12] Diagram demonstrating the differences between classical machine learning, representation learning and deep learning models. The shaded blocks are the learnable aspects of the associated models.

Representation learning can be incredibly powerful, especially when extracting the best factors of variation from the raw inputs requires a human level understanding of the data, which is something that typically cannot be easily encoded into a data prepossessing pipeline. An example hypothesis set for a typical representation learning model would be

$$\mathcal{H} = \{t_2 \circ t_1 : t_1 \in \mathcal{T}_1,\ t_2 \in \mathcal{T}_2\}.$$

The first learnable function can be seen as transforming the input into a feature vector, while the second learned transformation can be seen as transforming this feature into an output. Meanwhile, the hypotheses set of a typical deep learning model would take the form

$$\mathcal{H} = \{t_D \circ t_{D-1} \circ \cdots \circ t_2 \circ t_1 : t_i \in \mathcal{T}_i\ \forall i \in \mathbb{N}_{1:D}\}.$$

The hypotheses associated with a deep learning model can be seen as a hierarchy of representations. Each learned transformation is used to transform the previous simpler representation into a more complex/abstract representation of the data. D can be understood as the depth of the model. However, it should be mentioned that there is no consensus on how the depth of a deep learning model should be defined, nor how deep a representation learning model needs to be before it can be considered a deep learning model. Deep learning is thus a somewhat loose term, it can generally be thought of as a sub-field of machine learning encompassing models that attempt to learn a hierarchy of representations, a key characteristic of such models is the presence of many composed learned transformations [12]. The objective of taking this sequential approach of composing many learned transformations is to ideally produce more informative and complex features, features that could not feasibly be hand-designed and hence used by classical machine learning models.

2.3.2 Artificial Neural Networks (ANN)

Artificial neural networks are a type of representation learning model. The computational graphs of these types of models are designed to mimic the computation thought to occur in biological brains, these being networks of biological computational units called neurons. Although, one should be careful taking this connection between biological brains and ANNs too far, the connection is more akin to the relation between birds and planes, they are both able to perform a similar task, and indeed some aspects of a planes design could be seen to be inspired from birds, however, the actual mechanisms governing their operation are very different [21].
The basic computational unit associated with an ANNs is the artificial neuron (AN), it can be seen as a function/transformation AN that acts on some d dimensional input vector to produce a scalar output, i.e. $AN(\cdot;\ w,\ b,\ \sigma) : \mathbb{R}^d \to \mathbb{R}$. The first step of its computation is to take a weighted sum of the components of its presented input vector, this can be represented as $w^T x$, where w is a weight vector of equal dimension to the input. Typically a bias term is also added to this weighted sum $a = w^T x + b$, this offset weighted sum is called the pre-activation of the AN. This pre-activation is then taken and passed through some activation function $\sigma : \mathbb{R} \to \mathbb{R}$, this produces the post-activation $c = \sigma(a)$, which acts as the output of the AN. These computational steps are summarised by equation 16 [21].

$$AN(x;\ w,\ b,\ \sigma) = \sigma(w^T x + b) \tag{16}$$

An ANN is simply a network of these ANs, whereby the outputs from one AN can be used as a component for the input of another AN [21]. In general, each neuron has its own weight

vector, bias term and activation function. Naturally, neural networks can be represented using directed graphs, a networks associated directed graph is often referred to as its architecture. A set of nodes of this network are dedicated to components of the input, these nodes have no edges entering them. The other nodes in this network represent ANs, a subset of these nodes with no edges exiting them represent components of the output of the ANN. The nodes corresponding to ANs that do not produce components of the output are called hidden nodes, these nodes perform intermediate computations used to calculate the output, they are not directly observed when enumerating inputs and predicted outputs [12, 21].
An important subset of ANNs are feedforward neural networks. Their defining characteristic is that their architectures are directed acyclic graphs, this means that if one were to start on any node in the network, then it would be impossible to find a path following the directed edges that returned one to their original starting node. Intuitively, this means that information only flows forward in the network from input to output. Figure 6 shows an example of a non-feedforward ANN and a feedforward ANN.

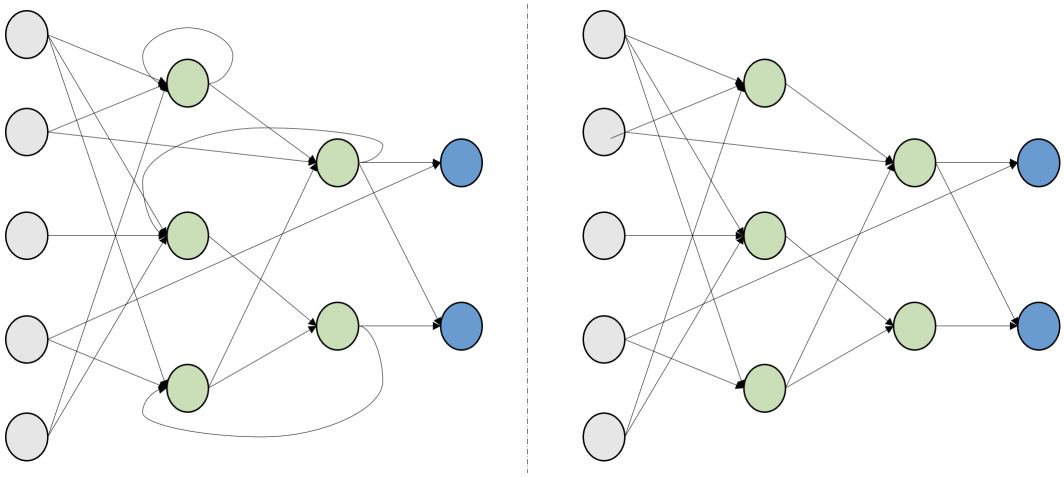

Figure 6: Diagram displaying examples of neural network architectures for a non-feedforward neural network (left) and a feedforward neural network (right). Input nodes are shaded grey, hidden nodes green and output nodes blue.

So far, ANNs have been discussed in terms of their associated hypothesis functions $ANN(\cdot; \mathcal{W}, \mathcal{B}, \tilde{\sigma}) : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{W}$, $\mathcal{B}$, $\tilde{\sigma}$ denote the tuples of weights, biases and activation functions associated with the ANs constituting the ANN. Naturally neural networks have a parametric representations, as such the hypothesis set can be recast in terms of the set of possible weight tuples, bias tuples and activation function tuples. Although, typically the activation function is not treated as learnable. There are exceptions to this however, an example of which being the PReLU activation function [21]. However, for the purposes of this report, activation functions will not be treated as learnable.
The process of learning the parameters associated with an ANN's ANs is equivalent to learning a transformation. The chaining of ANs in an ANN is thus equivalent to function composition of these learned transformations. This is why ANNs are considered representation learners. Furthermore, one could consider the longest path possible in a feedforward neural network's architecture as a measure of its depth, since increasing depth corresponds to more chained ANs and thus more composed learned transformations. Thus, ANNs have the capacity to be deep learning models, given their architectures constitute a sufficient depth.

2.3.3 Multilayer Perceptron (MLP)

Multilayer perceptrons are an important type of feedforward neural network. The nodes of a multilayer perceptron can be arranged into layers, such that nodes may only make connections with the nodes in layers directly proceeding. Furthermore, multilayer perceptrons demand that each node in a given layer form a connection with all nodes in the proceeding layer. For this reason, the multilayer perceptron is said to be composed of fully-connected layers, these layers are the most parameter dense type of layer possible. Figure 7 shows an example architecture for a multilayer perceptron.

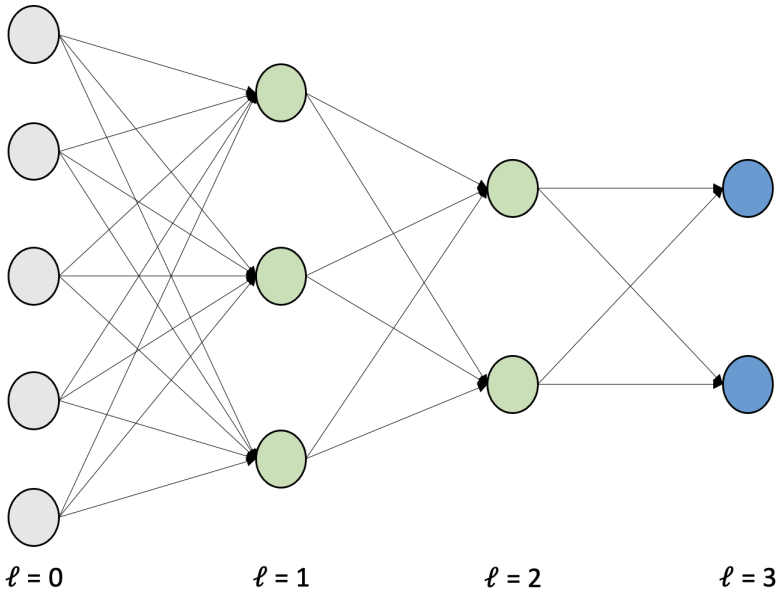

Figure 7: Diagram displaying an example neural network architectures for an MLP. Input nodes are shaded grey, hidden nodes green and output nodes blue.

The added structure associated with MLPs when compared to more general ANNs means that their hypotheses functions take a more specific and convenient to work with form. Let ℓ index the layers of the MLP, such that $\ell = 0$ corresponds to the input layer, $0 < \ell < D$ correspond to hidden layers and $\ell = D$ corresponds to the output layer, where D denotes the depth of the neural network. Additionally, let n_ℓ denote the number of hidden units in in layer ℓ. Lets now define the weight matrix $W^{(\ell)}$ associated with edges going from layer $\ell - 1$ to ℓ, such that $W_{ij}^{(\ell)}$ corresponds to the weight associated with the j'th post-activation of layer $\ell - 1$ in the weighted sum of the i'th pre-activation of layer ℓ's. Additionally, define the bias vector $b^{(\ell)}$ such that $b_i^{(\ell)}$ is the bias associated with the i'th pre-activation of layer ℓ. Let $a_i^{(\ell)}$, $c_i^{(\ell)}$ and $\sigma_i^{(l)}$ denote the pre-activation, post-activations and activation function for the i'th unit of layer ℓ, where the convention $c_i^{(0)} = x_i$ is used. Within this framework the previously discussed pre-activation and post-activation formulae become Equations 17 and 18 respectively.

$$a_i^{(\ell)} = \sum_{j=1}^{n_\ell} W_{ij}^{(\ell)} c_j^{(\ell-1)} + b_i^{(\ell)} \tag{17}$$

$$c_j^{(\ell)} = \sigma_i^{(l)}\left(a_i^{(\ell)}\right) \tag{18}$$

These equations can be vectorised by defining pre-activation $a^{(\ell)}$ and post-activation $a^{(\ell)}$ vectors of layer ℓ as the vectors whose components are $a_i^{(\ell)}$ $c_i^{(\ell)}$ respectively. Additionally, the vector function $\sigma^{(l)}$ can be defined such that $\sigma^{(l)}(a^{(\ell)})_i = \sigma_i^{(l)}(a_i^{(\ell)})$. This gives rise to the vectorised versions of the pre-activation and post-activation equation seen in Equations 19 and 20.

$$a^{(\ell)} = W^{(\ell)}c^{(\ell-1)} + b^{(\ell)} \tag{19}$$

$$c^{(\ell)} = \sigma^{(l)}\left(a^{(\ell)}\right) \tag{20}$$

Let $\omega^{(l)}(c^{(\ell-1)}) = W^{(\ell)}c^{(\ell-1)} + b$, this denoting the function that takes the previous layers post-activations to the current layers pre-activations. Under the ANN framework this is a learnable function, since the weights and biases are considered learnable parameters. Then, hypotheses of mlp's take the form:

$$\begin{aligned}
h(x) = c^{(D)} &= \sigma^{(D)}\left(a^{(D)}\right) = \sigma^{(D)}\left(\omega^{(D)}\left(c^{(D-1)}\right)\right) = \sigma^{(D)}\left(\omega^{(D)}\left(\sigma^{(D-1)}\left(a^{(D-1)}\right)\right)\right) \\
&= \cdots = \sigma^{(D)}\left(\omega^{(D)}\left(\sigma^{(D-1)}\left(\ldots\left(\omega^{(l+1)}\left(c^{(l)}\right)\right)\ldots\right)\right)\right) \\
&= \left(\sigma^{(D)} \circ \omega^{(D)} \circ \sigma^{(D-1)} \circ \omega^{(D-1)} \circ \cdots \circ \omega^{(\ell+1)}\right)(c^{(\ell)}) && (21) \\
&= \left(\sigma^{(D)} \circ \omega^{(D)} \circ \sigma^{(D-1)} \circ \omega^{(D-1)} \circ \cdots \circ \omega^{(\ell+1)} \circ \sigma^{(\ell)}\right)(a^{(\ell)}) && (22) \\
&= \left(\sigma^{(D)} \circ \omega^{(D)} \circ \sigma^{(D-1)} \circ \omega^{(D-1)} \circ \cdots \circ \sigma^{(1)} \circ \omega^{(1)}\right)(x) && (23)
\end{aligned}$$

Equation 23 demonstrates explicitly how an MLP is a type of representation learning model. Furthermore, Equation 21 demonstrates how a sufficiently deep MLP could be considered a deep learning model, since the hypothesis function can be considered to be a function of some intermediate representation $c^{(l)}$ in the hierarchy of representations $\left(c^{(\ell)}\right)_{\ell \in \mathbb{N}_{0:D}}$.

2.3.4 Backpropagation

ANNs are typically trained via gradient based optimisation. Such, optimisation methods require the calculation / estimation of gradients with respect to the ANN's learnable parameters. Backpropagation is an algorithm that allows for the efficient calculation of such gradients for feed-forward neural networks. The following will focus on MLPs specifically, although the procedure is easily generalised to general feedforward neural networks.
Let θ denote a general learnable hyperparameter vector, whose components constitute all weight matrix and bias vector entries for all layers. Suppose one wishes to to evaluate the gradient of the point wise loss function with respect the weight $W_{ij}^{(\ell)}$. It will be assumed that the loss function can be expressed explicitly in terms of the models predictions, i.e. $L(h(x;\theta), y)$. Then, given the function composition heavy form of equation 23 one would expect many applications of the chain rule. Fortunately, $W_{ij}^{(\ell)}$ only explicitly appears in the term $a_i^{(l)}$ and by extension only appears explicitly in $a^{(l)}$. Thus,

$$\frac{\partial L(h(x),\, y)}{\partial W_{ij}^{(\ell)}} = \left(\left(\nabla_{a^{(\ell)}} L\left(h\left(a^{(\ell)}\right),\, y\right)\right)^T \left(\frac{\partial a^{(\ell)}}{\partial a_i^{(\ell)}} \odot \frac{\partial a_i^{(l)}}{\partial W_{ij}^{(\ell)}}\right)\right) = \frac{\partial L\left(h\left(a^{(\ell)}\right),\, y\right)}{\partial a_i^{(\ell)}} \frac{\partial a_i^{(\ell)}}{\partial W_{ij}^{(\ell)}}.$$

The first factor in this product is called the sensitivity of the neural network to the i'th node in layer ℓ, it is explicitly defined in Equation 24 [**?**].

$$\delta_i^{(\ell)} \equiv \frac{\partial L\left(h\left(a^{(\ell)}\right),\, y\right)}{\partial a_i^{(\ell)}} \tag{24}$$

The second factor is easily evaluated using Equation 17 as $c_j^{(l)}$. Alternatively, had one instead have been investigating the gradient with respect to a bias, this second factor would simply evaluate to 1. Thus, the form gradient with respect to an arbitrary weight or bias is given by Equations 25 and 26 respectively [16].

$$\frac{\partial L\left(h(x),\, y\right)}{\partial W_{ij}^{(\ell)}} = \delta_i^{(\ell)} c_j^{(l)} \tag{25}$$

$$\frac{\partial L\left(h(x),\, y\right)}{\partial b_i^{(\ell)}} = \delta_i^{(\ell)} \tag{26}$$

In order to evaluate $h(x)$ the neural network already needs to calculate all the post-activations $c_j^{(l)}$ as intermediate results. since these are required for gradient calculations, the backpropagation algorithm records these intermediate values. It will be shown later that the pre-activations will also be necessary for gradient calculations, as such these will also be recorded. This step is called the forward-pass of the propagation algorithm [16].
The next step of the algorithm is calculating the node sensitivities. Unfortunately, assuming the loss and activation functions are differentiable, the only node sensitivities that are immediately calculable are the output node sensitivities [16].

$$\begin{aligned}\delta_i^{(D)} &= \frac{\partial L\left(h\left(a^{(D)}\right),\, y\right)}{\partial a_i^{(D)}} = \frac{\partial L\left(\sigma^{(D)}\left(a^{(D)}\right),\, y\right)}{\partial a_i^{(D)}} = \left(\nabla_{a^{(D)}} L\left(\sigma^{(D)}\left(a^{(D)}\right),\, y\right)\right)^T \left(\frac{\partial \sigma^{(D)}\left(a^{(D)}\right)}{\partial a_i^{(D)}}\right) \\ &= \frac{\partial L\left(\sigma^{(D)}\left(a^{(D)}\right),\, y\right)}{\partial \sigma_i^{(D)}\left(a_i^{(D)}\right)} \frac{\partial \sigma_i^{(D)}\left(a_i^{(D)}\right)}{\partial a_i^{(D)}}\end{aligned}$$

The above could be used to motivate a sensitivity calculation strategy that involves using sensitivities of higher layers to calculate the sensitivities of the nodes in lower levels. One could approach this problem by expressing the hypothesis function in Equation 24 in terms of the next layers pre-activation vector, exploitation of the chain rule would then generate the desired expression. Before this expression is derived let $\delta^{(\ell)}$ denote the sensitivity vector of layer ℓ, whose i'th component is equal to $\delta_i^{(\ell)}$. Then, the derivation of the necessary sensitivity expression is as follows [16]:

$$\begin{aligned}\delta_i^{(\ell)} &= \frac{\partial L\left(h\left(a^{(\ell+1)}\right),\, y\right)}{\partial a_i^{(\ell)}} = \left(\frac{\partial L\left(h\left(a^{(\ell+1)}\right),\, y\right)}{\partial a^{(\ell+1)}}\right)^T \left(\frac{\partial a^{(\ell+1)}}{\partial a_i^{(\ell)}}\right) \\ &= \left(\delta^{(\ell+1)}\right)^T \left(\frac{\partial \omega^{(l+1)}\left(c^{(\ell)}\right)}{\partial a_i^{(\ell)}}\right) = \left(\delta^{(\ell+1)}\right)^T \left(\frac{\partial \omega^{(l+1)}\left(\sigma^{(l)}\left(a^{(\ell)}\right)\right)}{\partial a_i^{(\ell)}}\right)\end{aligned}$$

$$= \left(\delta^{(\ell+1)}\right)^T \left(\frac{\partial}{\partial a_i^{(\ell)}} \left(W^{(l+1)} \sigma^{(l)}\left(a^{(\ell)}\right) + b^{(l+1)} \right) \right) = \left(\delta^{(\ell+1)}\right)^T \left(W^{(l+1)} \frac{\partial \sigma^{(l)}\left(a^{(\ell)}\right)}{\partial a_i^{(\ell)}} \right)$$

$$= \left(\delta^{(\ell+1)}\right)^T \left(W_{\cdot i}^{(l+1)} \frac{\partial \sigma_i^{(l)}\left(a_i^{(\ell)}\right)}{\partial a_i^{(\ell)}} \right) = \frac{\partial \sigma_i^{(l)}\left(a_i^{(\ell)}\right)}{\partial a_i^{(\ell)}} \left(\delta^{(\ell+1)}\right)^T \left(W_{\cdot i}^{(l+1)} \right)$$

$$= \frac{\partial \sigma_i^{(l)}\left(a_i^{(\ell)}\right)}{\partial a_i^{(\ell)}} \left(W_{\cdot i}^{(l+1)} \right)^T \left(\delta^{(\ell+1)}\right) = \frac{\partial \sigma_i^{(l)}\left(a_i^{(\ell)}\right)}{\partial a_i^{(\ell)}} \left(\left(W^{(l+1)} \right)_{i\cdot}^T \right)^T \left(\delta^{(\ell+1)}\right)$$

The vectorised version of this equation is shown in Equation 27. This provides a formula to calculate a layers sensitivity vector from the proceeding layers sensitivity vector. This gives rise to the backward pass stage of the propagation algorithm, where on move though the network from backwards starting at the end in order to calculate all the node sensitivities [16].

$$\delta^{(\ell)} = \nabla_{a^{(\ell)}} \sigma^{(l)}\left(a^{(\ell)}\right) \odot \left(\left(W^{(l+1)} \right)^T \delta^{(\ell+1)} \right) \tag{27}$$

With the network post-activations calculated from the forward pass and the network sensitivities calculated from the backward pass, Equations 25 and 25 can now be used to calculate derivatives of the point loss function with respect to weights and biases. These can then be used to calculate the gradient of a regularised empirical risk function, assuming that the regularisation function is differentiable. An optimizer can then take this gradient estimate and perform an update to the learnable parameters. These gradient based methods are typically iterative, i.e. this process will be repeated many times until some convergence criteria is met.

$$\nabla_\theta \hat{J}(\theta; \mathcal{D}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} \nabla_\theta L(h(x;\,\theta)) + \lambda \nabla_\theta \Lambda(\theta)$$

An alternative approach is to use the machinery provided by the backpropagation algorithm to calculate stochastic estimates of the gradients. The first step in this process is to randomly partition the training data into N_B approximately equally sized mini-batches $\mathcal{D}_{\text{train}} = \bigcup_{i=1}^{N_B} \mathcal{D}_i$, where N_B denotes the batch size. These mini-batches are then sequentially cycled through, first they are used to calculate an estimate of the empirical risk functions gradient:

$$G(\theta; \mathcal{D}_i) = \frac{1}{|\mathcal{D}_i|} \sum_{(x,y)\in\mathcal{D}_i} \nabla_\theta L(h(x;\,\theta)) + \lambda \frac{|\mathcal{D}_{\text{train}}|}{|\mathcal{D}_i|} \nabla_\theta \Lambda(\theta)$$

This gradient is then used by a gradient based optimizer to update the learnable parameters. This process is then repeated for the next mini-batch. The model is said to have trained for an epoch once a full cycle through all mini-batches has occurred. After each epoch the mini-batches are resampled and the process continues until some convergence criteria has been met. The use of stochastic gradients allows for more numerous gradient updates, more specifically a N_B fold increase in the number of gradient updates per set number of backpropagation applications. The downside is that the estimated gradients are less accurate. Empirically, it has been found that the benefit of more numerous gradient updates far outweighs this cost.

2.3.5 Optimizers

As was previously mentioned, neural networks are typically trained via gradient based optimisation, the gradients for which are calculated via the backpropagation algorithm. These methods work under the assumption that the loss function is sufficiently smooth with respect the learnable parameters, such that

$$\hat{R}(\theta + v,\ \mathcal{D}) \approx \hat{R}(\theta, x, y) + \nabla_\theta \hat{R}(\theta, \mathcal{D}) \cdot v, \text{ for } |v| << 1,$$

where v is some step in the parameter space. Thus, if one wishes to take a step in parameter space that maximises the decrease in empirical risk function it would be wise to select for the direction of the step vector to oppose the direction of the gradient. Furthermore, for more rapid convergence the magnitude of the step should be influenced by the magnitude of the gradient. The above motivates the stochastic Gradient Descent (SGD) algorithm shown in Algorithm 1 [12].

Algorithm 1 Stochastic Gradient Descent

Input: Dataset $\mathcal{D}$, initial parameter θ, number of epochs N_E, number of mini-batches N_B, learning rate η
  for $i \leftarrow 1$ to N_E **do**
    Randomly partition $\mathcal{D}$ into N_B mini-batches: $\mathcal{D}_1,\ \mathcal{D}_2,\ \ldots,\ \mathcal{D}_{N_B}$
    for $j \leftarrow 1$ to N_B **do**
      Use backpropagation to estimate stochastic gradients: $g \leftarrow G(\theta; \mathcal{D}_i)$
      Apply parameter update rule: $\theta \leftarrow \theta - \eta g$
    end for
  end for

A slight modification to the standard SGD algorithm is the SGD algorithm with momentum. This algorithm calculates an exponentially decaying moving average of the gradients, which has the effect of reinforcing updates to components of θ which act in the same direction, while penalising updates to components of θ whose gradient updates oscillate in direction. This is useful since oscillation is usually associated with overshooting a local minimum and then having to take a step backwards to readjust. The stochastic gradient descent algorithm with momentum is shown in Algorithm 2 [12].

Algorithm 2 Stochastic Gradient Descent with Momentum

Input: Dataset $\mathcal{D}$, initial parameter θ, number of epochs N_E, number of mini-batches N_B, learning rate η, initial velocity v, momentum parameter $\alpha \in [0, 1)$
  for $i \leftarrow 1$ to N_E **do**
    Randomly partition $\mathcal{D}$ into N_B mini-batches: $\mathcal{D}_1,\ \mathcal{D}_2,\ \ldots,\ \mathcal{D}_{N_B}$
    for $j \leftarrow 1$ to N_B **do**
      Use backpropagation to estimate stochastic gradients: $g \leftarrow G(\theta; \mathcal{D}_i)$
      Apply velocity update rule: $v \leftarrow \alpha v - \eta g$
      Apply parameter update rule: $\theta \leftarrow \theta + v$
    end for
  end for

Another popular class of optimizers are adaptive learning rate optimizers. These seek to tune/scale the learning rate parameter for different components of the learnable parameter. A popular example of such an optimizer is the RMSprop optimizer. This calculates a moving

average of the squared gradients, it then divides the learning rate by the square root of this moving average before each update. Updates to parameters that frequently encounter high magnitude gradient updates get penalised by this system, and vice versa for parameters that do not frequently encounter high magnitude gradient updates. The intuition behind this is similar to that of momentum, frequent high magnitude gradient updates are characteristic of oscillation, while gradients that gradually decrease are characteristic of a local minimum being approached, RMSprop pulls the update step towards parameter coordinates that are displaying the latter type of behaviour. It also allows for parameters which receive sparse updates, potentially due to the feature they are associated with being rare in the training data, to have an amplified update when they do receive a significant update. The RMSprop algorithm is displayed in Algorithm 3 [12].

Algorithm 3 RMSprop

Input: Dataset $\mathcal{D}$, initial parameter θ, number of epochs N_E, number of mini-batches N_B, learning rate η, decay rate ρ, division stability parameter $\delta_d \sim 10^{-6}$
Initialise $r = 0$
for $i \leftarrow 1$ to N_E **do**
Randomly partition $\mathcal{D}$ into N_B mini-batches: $\mathcal{D}_1$, $\mathcal{D}_2$, …, $\mathcal{D}_{N_B}$
for $j \leftarrow 1$ to N_B **do**
Use backpropagation to estimate stochastic gradients: $G(\theta; \mathcal{D}_i)$
Update moving average of squared gradients: $r \leftarrow \rho r + (1-\rho) g \odot g$
Apply parameter update rule: $\theta \leftarrow \theta - \frac{\eta}{\sqrt{\delta_d + r}} \odot g$
end for
end for

2.3.6 Activation Functions

Activation functions are continuous semi-differentiable functions [12]. They typically take on a simple form, this is both to aid computation and to comply the deep learning philosophy of building complex representations by sequentially composing simple learned transformations. The activation function of hidden nodes need to be non-linear, else the node becomes redundant, since the next layers pre-activation step is capable of performing this nodes computation by simply using a different set of weights. Differentiability is required for the backpropagation algorithm, however differentiability is typically not strictly demanded for the activation function. In the case of an activation function that is only semi-differentiable, the left derivative is taken by convention.
It is common practice to use the same activation function for all hidden nodes, meanwhile the output layer's activations are generally task-specific and different from the hidden layers. Two popular choice of hidden layer activation functions are the Rectified Linear Unit (ReLU) and the Exponential Linear Unit (ELU), their forms are given by Equations 28 and 29, Figure 8 displays plots of these activation functions.

$$\sigma_{\text{ReLU}}(a) = \left\{ \begin{array}{ll} 0 & a \leq 0 \\ a & a > 0 \end{array} \right\} \tag{28}$$

$$\sigma_{\text{ELU}}(a) = \left\{ \begin{array}{ll} \alpha\,(e^a - 1) & a \leq 0 \\ a & a > 0 \end{array} \right\} \tag{29}$$

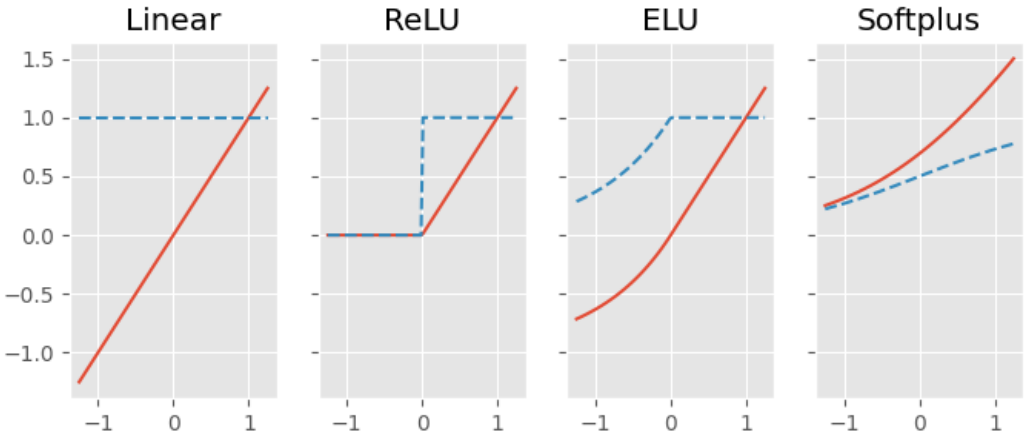

Figure 8: Diagram of Linear, ReLU, ELU ($\alpha = 1$) and Softplus activation functions (solid red line) and their derivatives (dotted blue line).

The ELU activation function can be viewed as a modified version of the ReLU activation function, the modification occurring for $a \leq 0$. The purpose of this modification is to mitigate the ReLU activation function's dying node problem. This being the phenomenon where an ANN updates its weights into a regime where the distribution of pre-activations going into a node favour negative values. Once this occurs it is very hard to get out of this regime due to the associated zero gradient of the ReLU activation function, which leads to a sensitivity of zero for the node, which prevents updates to the weights and biases entering the node. Once a node has died it will simply output zero for the remainder of traing, as many as half of the neurons in a ReLU hidden layer ANN can die over the course of training [21]. ELU solves this problem by have a non-zero gradient for all pre-activation values. The only downside is an increase in computational cost.

As mentioned previously, the output layer's activation functions are problem specific. For example, if one were performing regression and $\mathcal{Y} = \mathbb{R}$ then the linear activation would suffice. Alternatively if $\mathcal{Y} = \mathbb{R}^+$ the soft-plus activation may be preferable, its form is given by equation **??**. These activations can also be seen in Figure 8.

$$\sigma_{\text{softplus}}(a) = \log(1 + \exp(a)) \tag{30}$$

The most common and natural output activation function to use when performing multi-class classification is the Softmax activation function. This is because it produces post-activations in $(0, 1)$ and forces the summation of the output nodes' post-activations to be 1. Hence, when using this output activation the output nodes' post-activations can be interpreted as a categorical distribution of class membership probabilities $p(y|x, \theta)$. The Softmax activation does however have one subtlety, it requires the pre-activations of the entire output layer in order to be evaluated. Although, this breaks away from the formula of a feed-forward neural network, this is no cause for concern, since it is still trivial enough to calculate its gradient and by extension the sensitivities of the output nodes.

Its form is given by Equation 31.

$$\sigma_{\text{Softmax}}(a)_i = \frac{\exp(a_i)}{\sum_{j=1}^{n_D} \exp(a_j} \tag{31}$$

2.3.7 Convolutional Neural Networks

Convolutional neural networks are a very popular and powerful type of feed forward neural network. They modify the usual MLP model by adding additional convolutional and max-pooling layers to the beginning of the network. Convolutional layers have an associated dimensionality attribute which determines the layer's behaviour and expected input shape. The 2-Dimensional convolution operation can be thought of as applying a 3-dimensional filter over a 3-dimensional grid of post-activations from the previous layer, this grid having dimensions $H \times W \times C$. The application of this filter can be mathematically represented using a kernel $K_{i,j,k} \in \mathbb{R}^{f_h \times f_w, \times C}$, where f_W and f_H are the kernel size hyperparameters. The components of this kernel are the learnable weights of the convolutional layer. This kernel scans over the presented input grid of post-activations, at each step of the scan a pre-activation for the next layer is calculated by taking a weighted sum of the inputs within the kernels current receptive field and adding a bias term. The number of steps along the input grid that the kernel takes at each step of the scan is controlled by the stride size hyperparameters s_h and s_w. After the kernel has scanned through the entire input grid it will have produced a 2-dimensional grid of pre-activations, this is called a filter map. A convolutional layer will typically calculate many filter maps, each with its own unique learnable kernel and bias term. The previously used 3-dimensional kernel can thus be generalised to a 4 dimensional kernel, with the additional dimension used to index the kernels associated with different filter maps kernel $K_{i,j,k,l} \in \mathbb{R}^{f_h \times f_w, \times C \times N_f}$. The filter maps can then be stacked to produce a 3 dimensional grid of pre-activations, which after going through activation functions act as the inputs for a potential next convolutional layer. The above is summarised in equation 32 and displayed in Figure 9 [21].

$$a_{i,\ j,\ k} = b_k + \sum_{p=1}^{f_h} \sum_{q=1}^{f_w} \sum_{r=1}^{C} c_{(i-1)s_h+p,\ (j-1)s_w+q,\ r} K_{p,\ q,\ r,\ k} \tag{32}$$

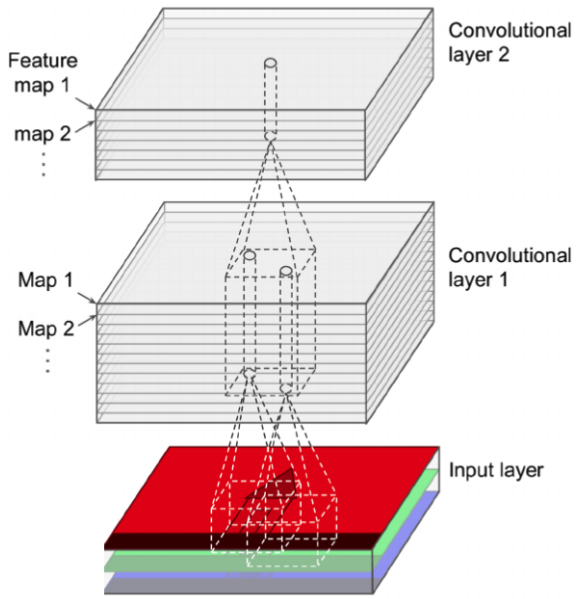

Figure 9: [21] Diagram showing the application of 2 convolutional layers to a 3-channel RGB input image.

The exact dimensions of the filter map will depend on the type of padding one uses. As a kernel scans over the input grid there may arise a scenario where the kernel is to be applied to a region too few nodes to fill the kernel. A common approach to dealing with this is stop calculating post-activations at this point, this is called valid padding. Another approach is zero padding, this replaces the missing nodes with zero values. There are many different choices of zero-padding, the most common is same zero-padding, this adds a sufficient number of dummy zeros such that a stride size one convolution will produce an output filter map of equal width and height to the input grid [21].
There are three reasons one would consider using convolutional layers:

- Sparsity: Each node in a convolutional layer is only connected to $f_h f_w C$ nodes from the previous layer via the kernel of its associated filter map. This is typically much lower than HWC, which would be the number of connections per node if input grid were to flattened and connected to a fully-connected layer. This greatly reduces the computation required to calculate pre-activations. Furthermore nodes deeper in a convolutional neural network are able to indirectly interact with a large portion of the input through the accumulation of the receptive fields of the nodes lower down in the network, this mitigating the downsides of a sparsity [12].

- Parameter sharing: The nodes of a given filter map share the same kernel and hence weights. This reduces the memory requirements of the model and can improve generalisations by mitigating overfitting [12].

- Equivariance to translation: Due to the parameter sharing between filters, translations to the inputs cause translations to the output filter maps. This makes convolutional layers a good choice for detecting specific patterns that occur within small windows of the input, but could occur anywhere in the input [12].

Another important type of layer used in convolutional neural networks is the pooling layer. Pooling layers apply some aggregating function over a set of nodes in the previous layer. Similar to the convolutional layer the pooling layer scans over the previous layer with a receptive field of some size $f_h \times f_w \times C$, the post-activations in its receptive field are then aggregated to produce a single summary statistic, this acts as the new post-activation. Aggregations are typically performed independently between different channels over the width and height of the previous layers grid of post-activations, thus, the number of filter maps is thus typically maintained. Conversely, the height and width of the input will see approximately s_h and s_w fold decreases respectively. This can greatly reduce the complexity of the model and hence reduce overfitting, there is also the added benefit of reduction in the computational requirements of the model [21]. An example of the action of a max pooling operation is shown in Figure 10.

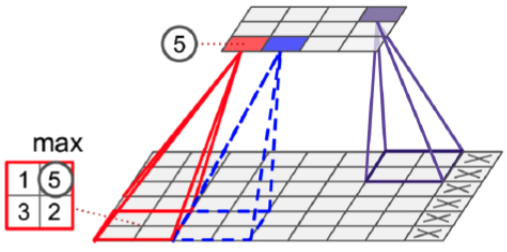

Figure 10: [21] Application of a valid padded max pooling operation, with a f_h, f_w, s_h, $s_w = 2$.

Additionally, aggregation can add some translation invariance to the model, small translations to an input may have negligible effect to the summary statistic associated with the receptive field of a node belonging to a pooling layer[12]. This can improve the data efficiency of the model if the exact location of feature matters little in comparison to the presence of a feature. The three most commonly used aggregation functions are the maximum, minimum and average.

2.3.8 Batch Normalization

A common problem encountered in deep learning is covariate shift. This occurs when a machine learning algorithm is trained on a dataset generated by some process $\mathbb{P}(\mathcal{D})$, but is tested / put into production on data generated from a different process $\mathbb{P}(\mathcal{D}^*)$. The machine learning algorithm was trained to minimise the Risk associated with data generated from $\mathbb{P}(\mathcal{D})$ and will not generally perform well on data generated from $\mathbb{P}(\mathcal{D}^*)$. This is immediately relevant to neural networks that train using stochastic gradients, the updates are relevant to a specific batch rather than the whole training data set, however it is hoped that the empirical distributions associated with the full data sets and the mini-batches constructed from it are sufficiently similar.
A related phenomenon, specific to deep learning is that of internal covariate shift. As has been discussed deep learning models learn a hierarchical representation of the data. Consider an intermediate representation, one could imagine the processing prior to this representation as a learned feature transformation and the processing after this representation as being a learned mapping from features to outputs. Suppose a gradient update causes a significant change to the parameters associated with the learned transformation used to get to the intermediate representation, then this would inevitably cause covariate shift in the intermediate representation. The consequence of this is that the learned intermediate feature to output mapping will have been trained on a different distribution of intermediate features.
Batch normalization seeks to mitigate these issues caused by large internal covariate shifts [24]. It is especially important for deep ANNs, since the risk for parameter changes upstream to cascade into internal covariate shifts downstream is far greater. This is because internal covariate shifts in one layer will cause internal covariate shifts in the next, thus compounding their effect as you go deeper into the network. Batch normalization produces a normalised intermediate $z_{(i)}$ representation using batch estimates for means μ_B and standard deviations σ_B. This is then scaled by some learned scaling vector γ and shifted by the learned shift vector β, giving rise to the batch normalised post-activations $\tilde{c}_{(i)}$. This process is summarised as follows, where δ_d denotes some division stability parameter [21]:

$$\mu_B = \frac{1}{M_B}\sum_{i=1}^{M_B} c_{(i)}$$

$$\sigma_B^2 = \frac{1}{M_B}\sum_{i=1}^{M_B}\left(c_{(i)} - \mu_B\right)^2$$

$$z_{(i)} = \frac{c_{(i)} - \mu_B}{\sqrt{\sigma_B^2 + \delta_d}}$$

$$\tilde{c}_{(i)} = \gamma \odot z_{(i)} + \beta$$

Batch-normalisation ensures that across every batch of training the distributions of specific intermediate representations stay approximately the same, this greatly improves the stability of training. Counterintuitively, although Batch normalization protects the ANN from large internal covariate shifts, it actually has the effect of causing minor covariate shifts every iteration of training. This is because the batch estimates for the mean and standard deviation vary between batches, thus leading to slight random shifts/scaling of the intermediate representation each iteration of training. This can be seen as a form of regularisation, it forces the network to be adaptive to slight changes in the intermediate representations distributions. Another way to think of this is as the network performing some slight random augmentation to the inputs depending on the random mini-batch they are being trained with [24].
The behaviour of a Batch normalization changes during testing, since the output of the ANN for a given output should not be dependent on the other inputs in the test set that it could be considered to be in a mini-batch with. Instead, during training a running average of a the relevant mean and standard deviation parameters is calculated and recorded. This running average then takes the place of the batch estimates during testing [21].

2.3.9 Dropout

Dropout is a popular method of regularisation used for training deep learning models. A dropout layer can be thought of a layer that takes a copy of the post-activations from the previous layer. At the beginning of each training step a dropout layer will go through each of the nodes in its layer and decide with probability p_d wether the node is going to be on or off, nodes that remain on simply pass on the post-activations they are fed, meanwhile a node that is turned off outputs zero. The effect of dropout layers is that at the start of each training step the layer that the dropout layer acts on randomly has an approximate proportion p_d of its nodes removed. It should be noted that during testing all nodes are turned on, this means that the layers that had dropout applied to them now have approximately $\frac{1}{1-p_d}$ more nodes on average than they did during training. To avoid internal covariate shift the weeights and biases of the layers feeding into dropout layers are multiplied by $(1 - p_d)$ during testing.
The ANN being trained under these circumstances, whereby neighbouring nodes are dropped out randomly every training step, mean that the nodes cannot become too co-dependant/co-adaptive on each other. The nodes need to learn how to effectively interact with all the nodes in the layer rather than a small handful, else their efficacy will be greatly reduced when these handful of neighbours are removed. This means the learned features associated with these nodes need to be more robust and informative over a wider range of training examples and features. This helps prevent overfitting, since a selection of nodes dedicated to learning a specific feature of a specific training example will no longer be effective in this type of network [25].
One can also see networks that utilise dropout training as a network that has been trained using an ensemble of network architectures. Suppose a network uses dropout training on N_d nodes, since these nodes can either be on or off during a given training iteration there are effectively 2^{N_d} different ANN architectures that can be used during training for a given iteration. Once it comes to testing the network has all nodes turned on. Since the network has been trained to perform well using any of these possible 2^{N_d} sub-networks, it follows that the final network can be seen as some sort of aggregated ensemble of these sub-networks [21, 25].

2.3.10 Neural Networks as Probabilistic Models

It is possible to frame ANNs as probabilistic models, and in doing so gain new intuitions about ANN design choices. This probabilistic interpretation has already been hinted at when the Softmax activation function was being discussed, where the output of an ANN using such an activation function can be interpreted as a categorical distribution of class memberships such that $h(x;\theta)_i = p(y = \hat{e}_i|x,\theta)$, where it is assumed the outputs are one-hot-encoded such that $\mathcal{Y} = \{0,1\}^{n_D}$. The following trick can be used to find an expression for $p(y = |x,\theta)$ explicitly in terms of $h(x;\theta)$:

$$p(y;x,\theta) = \prod_{j=1}^{n_D} p(y = \hat{e}_j;x,\theta)^{\mathbb{I}(y=\hat{e}_j)} = \prod_{j=1}^{n_D} p(y = \hat{e}_j;x,\theta)^{y_j} = \prod_{j=1}^{n_D} h(x;\theta)_j^{y_j}.$$

With an expression of for the class membership probability explicitly in terms of the neural network output one could be motivated to perform Maximum Likelihood Estimation (MLE) over the learnable parameters, or equivalently negative log likelihood minimisation. Let y_{ij} denote the j'th component of the i'th one-hot-encoded output vector in the dataset set, then the negative log-likelihood function is as follows:

$$\begin{aligned}
\text{NLL}(\theta;\mathcal{D}) &= -\log\left(p(\mathcal{D};\theta)\right) = -\log\left(\prod_{i=1}^{N} p(x_i,y_i;\theta)\right) = -\sum_{i=1}^{N} \log\left(p(x_i,y_i;\theta)\right) \\
&= -\sum_{i=1}^{N} \log\left(p(y_i|x_i;\theta)p(x_i;\theta)\right) = -\sum_{i=1}^{N} \log\left(p(y_i|x_i;\theta)p(x_i)\right) \\
&= -\sum_{i=1}^{N} \left(\log\left(p(y_i|x_i;\theta)\right) + \log\left(p(x_i)\right)\right) \\
&= -\sum_{i=1}^{N} \log\left(p(y_i|x_i;\theta)\right) - \sum_{i=1}^{N} \log\left(p(x_i)\right) \\
&= -\sum_{i=1}^{N} \log\left(\prod_{j=1}^{n_D} h(x;\theta)_j^{y_{ij}}\right) - \sum_{i=1}^{N} \log\left(p(x_i)\right) \\
&= -\sum_{i=1}^{N}\sum_{j=1}^{n_D} \log\left(h(x;\theta)_j^{y_{ij}}\right) - \sum_{i=1}^{N} \log\left(p(x_i)\right) \\
&= -\sum_{i=1}^{N}\sum_{j=1}^{n_D} y_{ij} \log\left(h(x;\theta)_j\right) - \sum_{i=1}^{N} \log\left(p(x_i)\right)
\end{aligned}$$

The second term has no θ dependence so can be ignored for the purposes of optimisation. Scaling the first term by some positive real number will also have no effect on the optimisation procedure. Thus MLE is equivalent to minimisation of the following function [15]:

$$\hat{R}(\theta,\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N}\left(-\sum_{j=1}^{n_D} y_{ij} \log\left(h(x;\theta)_j\right)\right)$$

This can be identified as the empirical risk function associated with the loss function presented in Equation 33. This loss function is called the categorical cross entropy [15].

$$L(h(x;\theta),y) = -\sum_{j=1}^{n_D} y_j \log\Big(h(x;\theta)_j\Big) \tag{33}$$

Alternatively, one could assume a prior on the parameters of the ANN and perform Maximum a Posteriori (MAP) estimation. This would be equivalent to minimisation of the negative log posterior.

$$\begin{aligned}
\text{NLP}(\theta;\mathcal{D}) &= -\log\left(\frac{p(\mathcal{D}|\theta)p(\theta)}{(\mathcal{D})}\right) \\
&= -\log(p(\mathcal{D}|\theta)) - \log(p(\theta)) + \log(p(\mathcal{D})) \\
&= NLL(\theta;\mathcal{D}) - \log(p(\theta)) + \log(p(\mathcal{D})) \\
&= \left(N\cdot\hat{R}(\theta,\mathcal{D}) - \sum_{i=1}^{N}\log\left(p\left(x_i\right)\right)\right) - \log(p(\theta)) + \log(p(\mathcal{D})) \\
&= (N\cdot\hat{R}(\theta,\mathcal{D}) - \log(p(\theta))) + \left(\log(p(\mathcal{D})) - \sum_{i=1}^{N}\log\left(p\left(x_i\right)\right)\right)
\end{aligned}$$

The second bracketed term has no θ dependence and can thus be ignored when optimising with respect to θ. Scaling the remaining term by a positive constant will also yield an equivalent objective function:

$$\hat{f}(\theta;\mathcal{D}) = \hat{R}(\theta,\mathcal{D}) - \frac{1}{N}\log(p(\theta))$$

This can be viewed as a regularised risk function. Thus, regularisation of a neural networks can be viewed as assuming some prior on the learnable parameters of the network and then subsequently performing MAP estimation [15]. Furthermore, one can consider MLE as a special case of MAP estimation for which one assumes an improper flat prior, this is mentioned to emphasise that ANNs can always be considered to assume some prior over their weights. Additionally, L1 and L2 regularisation correspond to isotropic laplacian and Gaussian priors respectively. This is shown as follows

$$\theta_i \overset{i.i.d}{\sim} \text{Laplace}\left(0,\frac{1}{N\lambda}\right):\ \log(p(\theta)) = \sum_{i}^{\dim(\theta)}\log\left(\frac{N\lambda}{2}\exp(-N\lambda|\theta_i|)\right) = -N\lambda|\theta|_1 + \dim(\theta)\log\left(\frac{N\lambda}{2}\right),$$

$$\theta_i \overset{i.i.d}{\sim} \mathcal{N}\left(0,\frac{1}{2N\lambda}\right):\ \log(p(\theta)) = \sum_{i}^{\dim(\theta)}\log\left(\sqrt{\frac{N\lambda}{\pi}}\exp(-N\lambda\theta_i^2)\right) = -N\lambda|\theta|_2^2 + \dim(\theta)\log\left(\sqrt{\frac{N\lambda}{\pi}}\right),$$

The additive constants at the end of these expressions do not depend on the value θ and can thus be ignored. Combining this with the previously derived objective function for MAP estimation yields the familiar form of L1 and L2 regularisation. This brings a new perspective on these types of regularisation, it was previously seen that L1 and L2 regularisation were equivalent to constrained optimisation on hyper-cubes and hyper-spheres respectively, with higher regularisation hyper parameters decreasing the volume of these constraining manifolds. This can now be understood in terms of a stricter prior, as λ increases the scale parameter of these priors decreases, thus localising the learnable parameters to a smaller volume. Large amounts of data/evidence will then be required in order for the posterior to deviate significantly from these strong priors, hence the regularising effect.

2.3.11 Bayesian Neural Networks

So far neural networks have been framed within a probabilistic framework to the extent where they can now model aleatoric uncertainty. This form of uncertainty is essentially the type of uncertainty that is built into a model, i.e. assuming a model is an accurate description of reality then for a given input x the distribution of classes follows a categorical distribution $p(y|x;\theta)$. Of course, this is somewhat disingenuous in most cases, it is very unlikely that a machine learning practitioner truly believes that some specific converged upon θ, wether arrived upon via MLE and MAP estimation, gives rise to a model that perfectly describes the true data generation process. This is especially true in deep learning, where simply restating the training algorithm will almost certainly lead to a different converged upon θ due to θ's random initialisation. The machine learning practitioner almost certainly holds some uncertainty in their belief that the converged upon θ is truly reflective of the data generation process. This is exactly the type of uncertainty that Bayesian inference is designed to handle, this uncertainty over the trained network parameters of the model is called epistemic uncertainty, it is quantified by the posterior distribution over the network parameters.
Bayesian Neural Networks (BNNs) attempt to tackle this problem of calculating the weight posterior. This being in contrast to the conventional approach of implicitly assuming some prior over the network parameters and then finding a point estimate of its network parameters' posterior. These differences are visualised in Figure 11.

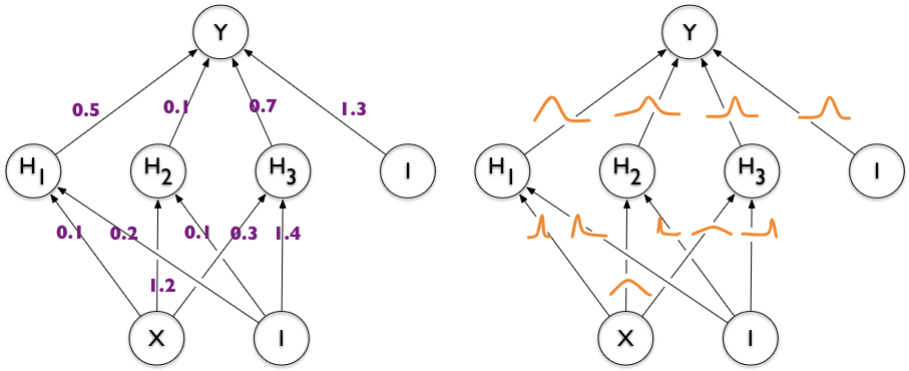

Figure 11: [15] Visualisation of the difference in parameter inference performed by standard ANNs (left) versus BNNs (right). Standard ANNs perform point estimation of the network's parameters, while BNNs attempt to find the posterior over parameters.

The weight posterior and marginal likelihood take the forms:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})},$$
$$p(\mathcal{D}) = \int_{\theta\in\Omega_\theta} p(\mathcal{D}|\theta)p(\theta)d\theta.$$

Unfortunately the integral associated with the marginal likelihood is computationally intractable. This is because even the most shallow neural networks contains thousands of parameters, i.e. in general $|\Omega_\theta| >> 10^3$. Instead, variational inference can be used to approximate the posterior [15]. This involves assuming some family of variational distributions Q, it assumed that this family is parametric such that $Q = \{q(\cdot;\ \phi) : \phi \in \omega_\phi)\}$. The network parameters, i.e. the weights and biases, are no longer the learnable

parameters of this network, in fact these are simply random variables. Instead, the variational parameters controlling the distribution of the network parameters are the learnable parameters.
Recall, optimisation over a family of variational distributions can be accomplished via the variational free energy objective function. This objective function was first displayed in Equation 7, although it is restated below to aid readability:

$$\mathcal{F}(\phi;\mathcal{D}) = \mathbb{E}_{\theta\sim q(\theta;\ \phi)}\left(-\log\left(p(\mathcal{D}|\theta)\right)\right) + D_{KL}(q(\theta;\ \phi)\|p(\theta))$$

First, the variational free energy should be identified as a regularised risk function, such that its associated loss function can be found:

$$\begin{aligned}
\underset{\phi\in\Omega_\phi}{\operatorname{argmin}}\ \mathcal{F}(\phi;\mathcal{D}) &= \underset{\phi\in\Omega_\phi}{\operatorname{argmin}}\Big(\mathbb{E}_{\theta\sim q(\theta;\ \phi)}\left(-\log\left(p(\mathcal{D}|\theta)\right)\right) + D_{KL}(q(\theta;\ \phi)\|p(\theta))\Big) \\
&= \underset{\phi\in\Omega_\phi}{\operatorname{argmin}}\left(\mathbb{E}_{\theta\sim q(\theta;\ \phi)}\left(-\log\left(\prod_{i=1}^{N} p(y_i|x_i,\theta)p(x_i)\right)\right) + D_{KL}(q(\theta;\ \phi)\|p(\theta))\right) \\
&= \underset{\phi\in\Omega_\phi}{\operatorname{argmin}}\left(\mathbb{E}_{\theta\sim q(\theta;\ \phi)}\left(-\sum_{i=1}^{N}\log\left(p(y_i|x_i,\theta)p(x_i)\right)\right) + D_{KL}(q(\theta;\ \phi)\|p(\theta))\right) \\
&= \underset{\phi\in\Omega_\phi}{\operatorname{argmin}}\left(\mathbb{E}_{\theta\sim q(\theta;\ \phi)}\left(-\sum_{i=1}^{N}\log\left(p(y_i|x_i,\theta)\right)\right) + D_{KL}(q(\theta;\ \phi)\|p(\theta))\right) \\
&= \underset{\phi\in\Omega_\phi}{\operatorname{argmin}}\left(\sum_{i=1}^{N}\mathbb{E}_{\theta\sim q(\theta;\ \phi)}\left(-\log\left(p(y_i|x_i,\theta)\right)\right) + D_{KL}(q(\theta;\ \phi)\|p(\theta))\right) \\
&= \underset{\phi\in\Omega_\phi}{\operatorname{argmin}}\left(\frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{\theta\sim q(\theta;\ \phi)}\left(-\log\left(p(y_i|x_i,\theta)\right)\right) + \frac{1}{N}D_{KL}(q(\theta;\ \phi)\|p(\theta))\right)
\end{aligned}$$

The above is clearly identified as a regularised risk function, the associated point-wise loss function of this regularised risk function is the expected categorical cross-entropy under the variational distribution. The point-wise loss and regularised risk functions are given by Equations 34 and 35

$$L(h(x;\phi),y) = \sum_{j=1}^{n_D}\mathbb{E}_{\theta\sim q(\theta;\ \phi)}\Big(-y_j\log\Big(h(x;\theta)_j\Big)\Big) \tag{34}$$

$$\hat{J}(\phi;\ \mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{n_D}\mathbb{E}_{\theta\sim q(\theta;\ \phi)}\Big(-y_{ij}\log\Big(h(x_i;\theta)_j\Big)\Big) + \frac{1}{N}D_{KL}(q(\theta;\ \phi)\|p(\theta)) \tag{35}$$

Taking expectations with respect to θ gives rise to the same problems that were encountered previously, the associated integral is very high dimensional and as such is computationally intractable. Evaluation of Equation 35 is instead performed using Monte-Carlo estimation [15], before this can be performed lets first rewrite the equation.

$$\hat{J}(\phi;\ \mathcal{D}) = \mathbb{E}_{\theta\sim q(\theta;\ \phi)}\left(\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{n_D}-y_{ij}\log\Big(h(x_i;\theta)_j\Big) + \frac{1}{N}\log\left(\frac{q(\theta;\ \phi)}{p(\theta)}\right)\right) = \mathbb{E}_{\theta\sim q(\theta;\ \phi)}\Big(\hat{j}(\theta,\phi,\mathcal{D})\Big)$$

$$\hat{j}(\theta,\phi,\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{n_D}-y_{ij}\log\Big(h(x_i;\theta)_j\Big) + \frac{1}{N}\log\left(\frac{q(\theta;\ \phi)}{p(\theta)}\right)$$

Thus a Monte-Carlo estimate for J could be generated by sampling N_M samples $\theta^{(i)}$ from the variational distribution and then taking the sample mean of their evaluations on $\hat{j}$, this is demonstrated as follows

$$\hat{J}(\phi;\ \mathcal{D})^{(MC)} = \frac{1}{N_M}\sum_i \hat{j}(\theta^{(i)}, \phi;\ \mathcal{D}).$$

To perform backpropagation the gradient of J will have to be calculated. In order to make progress one needs to restrict the way in which θ and ϕ can be related. More specifically, it is demanded that the stochastic part of their relationship be governed by some parameterless noise $\epsilon \sim q(\epsilon)$, such that $\theta = t(\phi,\ \epsilon)$. Under these conditions the expectation can be reparameterised in terms of parameterless noise, which in turn allows derivatives with respect to ϕ to be taken inside the expectation [15].

$$\nabla_\phi \hat{J}(\phi;\ \mathcal{D}) = \nabla_\phi \mathbb{E}_{\theta\sim q(\theta;\ \phi)}\left(\hat{j}(\theta,\phi,\mathcal{D})\right) = \nabla_\phi \mathbb{E}_{\epsilon\sim q(\epsilon)}\left(\hat{j}(t(\phi,\epsilon),\phi,\mathcal{D})\right) = \mathbb{E}_{\epsilon\sim q(\epsilon)}\left(\nabla_\phi \hat{j}(t(\phi,\epsilon),\phi,\mathcal{D})\right)$$

With the derivative now under the expectation, one can calculate Monte-Carlo estimates for the gradients, this is demonstrated as follows:

$$\nabla_\phi \hat{J}(\phi;\ \mathcal{D})^{(MC)} = \frac{1}{N_M}\sum_i \nabla_\phi \hat{j}(t(\phi,\epsilon^{(i)}),\phi;\ \mathcal{D}).$$

The first term of j is simply the empirical risk associated with the categorical cross-entropy loss function L. Since the derivative will distribute over the summations in this risk function, on need only be able to calculate the derivatives of the categorical cross entropy. Backpropagation allows for these derivatives to be calculated with respect to $\theta = t(\phi,\ \epsilon)$. Assuming t is differentiable, then an application of the chain rule will enable these derivatives to be calculated with respect to ϕ:

$$\nabla_\phi L(h(x;\ \theta), y) = \nabla_\phi L(h(x;\ t(\phi,\epsilon)), y_i) = \nabla_\theta L(h(x;\ t(\phi,\epsilon)), y_i) \odot \nabla_\phi t(\phi,\ \epsilon))$$

The second term of j will be a simple enough to evaluate, so long as the family of variational distributions takes a simple form, for example a subset of the exponential family of distributions.
The above allows for full batch optimisation to be performed, although, as previously discussed deep learning models are best trained by mini-batch optimisation. Up until now the variational free energy objective was divided by N, this was to make the link between this function and a regularised risk function explicit. However, this convention will now be dropped for succinctness, this gives rise to the following batched objective function.

$$J_B(\phi;\ \mathcal{D}) = \mathbb{E}_{\theta\sim q(\theta;\ \phi)}\left(j_B(\theta,\phi,\mathcal{D})\right)$$
$$j_B(\theta,\phi,\mathcal{D}) = \sum_{i=1}^{N}\sum_{j=1}^{n_D} -y_{ij}\log\left(h(x_i;\theta)_j\right) + \log\left(\frac{q(\theta;\ \phi)}{p(\theta)}\right)$$

Suppose the dataset is randomly partitioned into N_B mini-batches $\mathcal{D}_1, \mathcal{D}_2 \ldots \mathcal{D}_{N_B}$. Furthermore let $\pi \in [0,1]^{N_B}$ such that $\sum_{i=1}^{N_B} \pi_i = 1$. Then, it can be easily shown that the relations between Equations 36, 37 and 38 hold [15].

$$J_B(\phi;\ \mathcal{D}) = \sum_{i=1}^{N_B} J_B^{\pi_i}(\phi;\ \mathcal{D}_i) \tag{36}$$

$$J_B^{\pi_i}(\phi;\ \mathcal{D}_i) = \mathbb{E}_{\theta \sim q(\theta;\ \phi)}\left(j_B^{\pi_i}(\theta, \phi, \mathcal{D}_i)\right) \tag{37}$$

$$j_B^{\pi_i}(\theta, \phi, \mathcal{D}_i) = \sum_{(x,y)\in\mathcal{D}_i} \sum_{j=1}^{n_D} -y_j \log\left(h(x;\theta)_j\right) + \pi_i \log\left(\frac{q(\theta;\ \phi)}{p(\theta)}\right) \tag{38}$$

Thus, one way to mini-batch enable the optimisation of a BNN is to sequentially cycle through the mini-batches, at each step using the optimisation of $J_B^{\pi_i}(\phi;\ \mathcal{D}_i)$ to make a weight update to ϕ. Again, subject to θ being parameterisable in terms of parameterless noise, one could then calculate Monte-Carlo estimates of $J_B^{\pi_i}$ and $\nabla_\phi J_B^{\pi_i}$ by averaging the samples of $j_B^{\pi_i\ (i)}$ and $\nabla_\phi j_B^{\pi_i\ (i)}$ associated with draws from $\epsilon^(i) \sim q(\epsilon)$. The most commonly used regulariser weighting scheme is a uniform one, where $\pi_i = \frac{1}{N_B}$. Another popular scheme is $\pi_i = \frac{2^{N_B - i}}{2^{N_B}-1}$, this can be seen to regularise the earlier mini-batches more than the later mini-batches, which can be understood as encouraging the network to wait until it has seen more data before making large weight updates [15].
This above approach is a little different in comparison to the usual approach of finding an associated regularised risk function and simply substituting in a mini-batch. The difference can be understood by the subtlety that in order for variational inference to work, a total effective complexity penalty of exactly $D_{KL}(q(\theta;\phi)\|p(\theta))$ needs to be applied per epoch, the previous method would have essentially applied this amount of penalty each update, thus over-regularising the optimisation, resulting in a converged upon result that would behave as if a stronger prior had been used.
Once the network is trained, such that a set of variational parameters ϕ^* have been found, the BNN makes predictions by taking the expected network prediction over the inferred variational posterior. This is called the Variational Predictive Distribution (VPD) and it is displayed in Equation 39. In practice the VPD is evaluated using Monte-Carlo estimation. The VPD can be seen as a variational approximation of the Bayesian Model Average (BMA), this is shown in Equation 40.

$$\hat{p}(y|x, \phi^*) = \mathbb{E}_{\theta \sim q(\theta|\phi^*)}(h(x;\ \theta)) \tag{39}$$

$$p(y|x, \mathcal{D}) = \mathbb{E}_{\theta \sim p(\theta|\mathcal{D})}(h(x;\ \theta)) \tag{40}$$

2.3.12 A Bayesian Perspective on Generalisations

Historically statisticians and machine learning practitioners have used many heuristics to guide their model design choices. One such heuristic is the “one in ten” rule, sating that in order to avoid overfitting one should have at least 10 data points per parameter in a model [16]. This heuristic fails spectacularly when applied to deep learning, where it is not uncommon for there to be orders of magnitude more parameters than data points, yet these models are often able to achieve state of the art performance. An example of this is the VGG-16, a model with $\approx 138B$ parameters [26]. This CNN is a top-performing model, originally trained for image classification on the famous image database ImageNet,

consisting of $\approx 15M$ training examples over $\approx 22k$ classes [27]. Clearly, simple parameter counting is a poor predictor of a models ability to generalise. The unusual generalisation behaviour exhibited by ANNs can be understood if one instead takes a Bayesian perspective on generalisations. This perspective argues that generalisation depends on two properties, the models support and inductive biases.
The support of a model $\mathcal{M}$ can be understood as the set of datasets that can be explained by the model, mathematically this could be represented as $\{\mathcal{D} : p(\mathcal{D}|\mathcal{M}) > 0\}$. The support as a predictor of generalisation serves a similar purpose to parameter counting, since counting parameters is considered to give a proxy for model complexity. The more complex a model is the larger the variety of relationships it can model, and as such the larger the variety of datasets it can generate/describe, hence the larger its support [28].
The inductive biases of a model can be understood as the models prior relative consistency with different datasets prior to training. This is encoded by the marginal likelihood $p(\mathcal{D}|\mathcal{M})$, this being the expected likelihood that a dataset would be generated when sampling hypotheses from the prior. Suppose a model has a high marginal likelihood for datasets typical of a specific class of problem. This indicates the existence of probable hypotheses, under the prior, which are highly consistent with the data typical of this class of problem. Typically, large deviations from the prior demand large amounts of data. The fact that the prior already has a propensity for hypothesis consistent with the data means that a model applied to such a class of problem will be far more data efficient in its objective to select a model consistent with the data. Within machine learning bias usually caries a negative connotation, the above illustrates that when a models inductive biases are well calibrated to problem, then the presence of inductive biases can be very beneficial. This perspective reveals an insight that simple parameter counting is not able to describe - a models ability to generalise depends on its suitability to the problem it is being applied to [28].
Consider the comparison in performance between an MLP, CNN and simple linear model, each having a similar number of parameters, when applied to an image recognition task. MLPs have very large supports and can model a wide variety of phenomenon, in fact a single hidden layer MLP of arbitrary width can be shown to be a universal approximator [12]. Unfortunately they do not have very strong inductive biases favouring any particular type of problem's associated datasets, MLPs are equally well suited to a wide variety of problems. Conversely, CNNs were specifically designed with inspiration from the human visual cortex, it is no surprise that these models have inductive biases which are well calibrated toward image recognition problems. Additionally they do not sacrifice much in terms of support, as discussed previously the deeper nodes of a CNN can indirectly interact with a large effective receptive field over the input, and as such they are capable of modelling a similar variety of phenomenon compared to MLPs. The simple linear model has the opposite problem to the MLP, it has strong inductive biases, however its support does not support generation of datasets associated with image recognition problems. Thus, the large support and well calibrated inductive biases of the CNN will allow it be more data efficient while simultaneously converging to a high quality solution for this problem. The above comparisons are summarised in Figure 12.

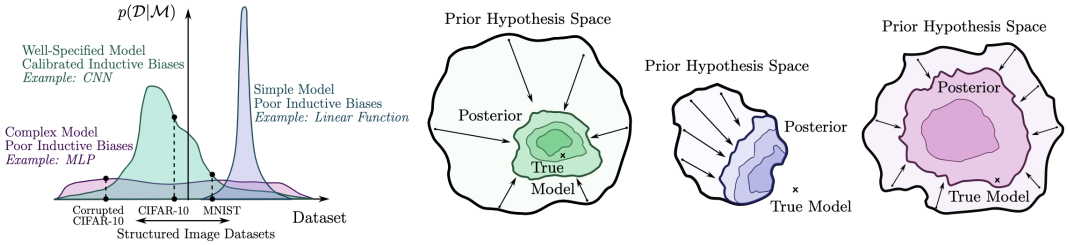

Figure 12: [28] A qualitative plot of the effects different model's support and inductive biases induce when applied to an image recognition problem. Green shading is associated with the CNN, blue with the simple linear model and pink with the MLP. The CNN is able to converge fastest and to a high quality solution, The simple linear model converges quickly, but to a low quality solution and the MLP converges slowly to a high quality solution.

One can now view neural network design choices, which previously may have seemed quite arbitrary, as methods of calibrating the model's inductive biases to a specific problem. Examples of design choices being network architecture, layer types, optimizer type, activation functions, regularisation type, ect. This means, counterintuitively, opting for a more "complex" model component, e.g. choosing an ELU activation over a ReLU activation, could potentially improve generalisation and reduce overfitting.

Although the design choices for ANNs are seemingly endless, ANNs do all share one issue in common, they are typically under-specified by the training data [28]. Under-specification referring to the scenario where there are many parameter values which are consistent with data, leading to a hypothesis set containing many high performing models for a given dataset. The problem with under-specified models is that the choice of parameter, when performing point estimation, is somewhat arbitrary. Suppose there is a choice between two parameter vectors which are well separated in parameter space, such that the associated hypotheses are functionally dissimilar. Now suppose these parameters give rise to very similar performance on the training data, for example suppose one hypothesis performs 0.01% better than the other model. Choosing one hypothesis over another based on such a marginal difference in performance seems foolish, especially when one considers that the empirical risk function is only an estimate of the true risk function. The difference in functional form of these hypotheses could lead to one hypothesis performing well on inputs that the other hypothesis performs poorly on. Surely a better approach would be to somehow aggregate their predictions to form a better prediction. This is exactly what the BMA of model with respect to its posterior over parameters is attempting to do [28].

An ANN that attempts to estimate the BMA has already been encountered, this being a BNN with its VPD. Unfortunately, a BNN's VPD is usually a poor estimate of the BMA. This is because typical choices of variational distribution families are too simple to capture the true posterior well. For example, a typical choice of variational posterior is the isotropic Gaussian, its popularity being due to the associated reduction in computational cost from this mean-filed variational family of distributions. The problem with using a unimodal variational family of distributions is that it will only be able to accurately capture a single mode of the true posterior [29]. However, the benefits associated with using the BMA are that a large variety, in terms of functional form, of solutions are aggregated to hopefully give a better solution. Thus, if the VPD is to confer a significant benefit, the mode approximated by the selected variational distribution needs to encompass a wide variety of solutions. Generally this achieved when the approximated posterior mode is flatter, leading to a wider region of Ω_θ being given significant probabilistic contributions to the VPD,

which ideally leads to a wide variety of high performance hypothesis in $\mathcal{H}$ being aggregated. It should also be noted that BNNs have a blessing of dimensionality in this case, since flat region of approximately constant scale in each dimension will encompass an exponentially larger volume as the dimensionality of the parameter space increases [28]. Another popular method of aggregating a diverse set of high performing solutions is that of deep ensembles. ANNs have highly complex non-convex loss functions, a simplified diagram of this is shown in Figure 13. This means that minimisation through gradient based methods will typically cause convergence to a one of many highly performing local minima of the loss function. A consequence of this is that the random initialisation of neural network parameters will lead to drastically different converged upon solutions. Additionally, Each of these solutions is a top performing solution within a local region of the parameter space. Deep ensembles exploit these properties by randomly initialising, training, and then aggregating the predictions of M ANNs. The literature indicates that this approach often outperforms that of a comparable BNN [15]. The downside of Deep ensembles is that their computational cost is M times greater than an equivalent ANN.

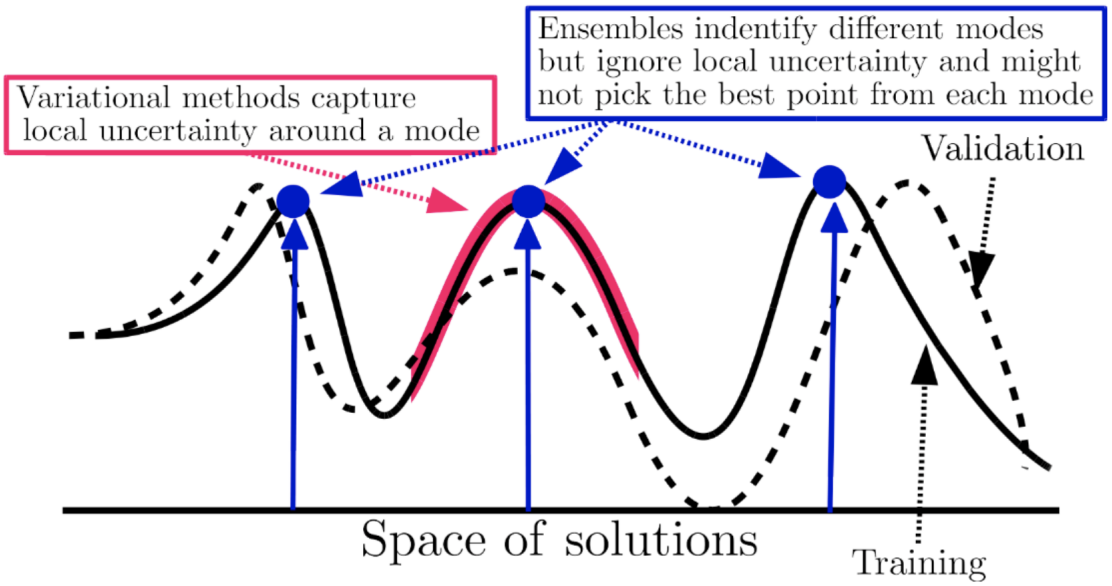

Figure 13: [29] Diagram depicting the negative empirical risk function associated with a training dataset (solid line) and validation dataset (dashed line). For simplicity, an improper flat prior is assumed such that the negative empirical risk function and log-likelihood function coincide, which in turn leads to the log posterior being proportional to the negative empirical risk function. A deep ensemble aggregates the predictions from networks which converge to the peaks of different modes, while a BNNs aggregation is governed by a local estimate about a single node.

3 Methodology

3.1 Dataset Description

The objective of this work was to investigate the relative efficacies of comparable non-Bayesian and Bayesian ANNs applied to the task of audio classification. The null hypothesis being that ANNs will perform equally if not better than BNNs. The alternative hypothesis that BNNs will will outperform the ANNs is motivated by their utilisation of the VPD. The advantage of utilising a BMA approximation in audio classification problems arises due to the propensity of such problems to cause under specification of trained ANNs. The dataset that was used to investigate this hypothesis was the "ESC-50: Dataset for Environmental Sound Classification" dataset [30]. This is a very popular dataset within the field of audio classification and it was created specifically with reproducibility in mind, with the dataset being pre-partitioned into 5 cross-validation folds [30].
The dataset consists of 2000 audio recordings, this comprising of 40 observations across 50 different classes, the dataset is thus balanced. Furthermore the 50 classes can be further sub-divided into one of 5 categories, each category has 10 associated classes. A summary of the class and class categories can be seen in Table 2, additionally Figure 14 shows an example waveform from each of these classes. As can be seen from this table, this dataset contains a wide variety of environmental sounds. Pairing this with the large number of classes and relatively small number of observations per class, and it is easy to understand that this is a challenging dataset to work with. In fact, when presented with this dataset humans are only able to acheive 81.3% accuracy [30].

Animal	Landscape and Water	Human	Urban Interior	Urban Exterior
Dog	Rain	Crying Baby	Door Wood Knock	Helicopter
Rooster	Sea Waves	Sneezing	Mouse Click	Chainsaw
Pig	Crackling Fire	Clapping	Keyboard Typing	Siren
Cow	Crickets	Breathing	Door Wood Creaks	Car Horn
Frog	Chirping Birds	Coughing	Can Opening	Engine
Cat	Water Drops	Footsteps	Washing Machine	Train
Hen	Wind	Laughing	Vacuum Cleaner	Church Bells
Insects	Pouring Water	Brushing Teeth	Clock Alarm	Airplane
Sheep	Toilet Flush	Snoring	Clock Tick	Fireworks
Crow	Thunderstorm	Drinking Sipping	Glass Breaking	Hand Saw

Table 2: Different classes of audio present in the ESC-50 dataset, with their associated categories indicted by the column titles.

In terms of the technical details pertaining to the dataset. Each audio recording had a sample rate of $s = 44.1$ kHz, a bit depth of $\rho = 16$ and a clip duration of $T = 5$ s. This giving rise to a high dimensional input space $\mathcal{X} \subseteq \mathbb{Z}^d_{(-2^{\rho-1}):(2^{\rho-1}-1)}$, where d$= f_S T = 220500$, and a modestly sized dataset of size ≈ 1 GB. However, as a first pre-processing step the inputs were divided by $2^{\rho-1}$ such that $\mathcal{X} \subset [-1, 1)^d \subset \mathbb{R}^d$. The storage size of this dataset is relevant once one begins considering data augmentation. Most commercial laptops / computers have memory of order ∼8 GB, thus a significant increase in the size of the dataset caused by data augmentation will in general make manipulations with this dataset cumbersome or even impossible. For this reason, alongside the speed benefits associated with GPU accelerated training, the computation of this investigation was facilitated by a GPU cluster provided by Imperial College London's Mathematics Department.

Figure 14: Plot displaying example waveforms form each of the 50 classes, the columns of this plot grid are associated with class categories, see Table 2 for further information.

3.2 Feature Extraction

3.2.1 The Fourier Transform (FT)

An initial inspection of of Figure 14 demonstrates the wide variety of signals present in the ESC-50 dataset. Unfortunately, even a human would find it difficult to discriminate between the classes given nothing more than a printout of the audio's waveform. As with many machine learning problems it can be useful to look towards humans for inspiration, after all humans have evolved to excel at tasks pertaining to navigation and manipulation of their environment, classifying sound being one of these. Between a mechanical sound wave entering the human ear and its associated electrical signal being transmitted via the auditory nerve to the brain, there are many of what can be considered to be biological pre-processing steps. Importantly hair cells embedded within the cochlea can be seen to be performing the task of frequency decomposition, with different hair cells resonating at different frequencies [31].
Frequency decomposition of a signal can be performed via the Fourier transform F. The Fourier transform is able to take some integrable signal $s(t)$ and move it into a frequency domain representation $\tilde{s}(f)$. Furthermore, the Fourier transformation is bijective and has an associated inverse. This is summarised in Equations 41 and 42 [32].

$$\tilde{s}(f) = F(s)(f) = \int_{\mathbb{R}} s(t)\exp(-i2\pi f t)dt \tag{41}$$

$$s(t) = F^{-1}(\tilde{s}) = \frac{1}{2\pi}\int_{\mathbb{R}} \tilde{s}(f)\exp(i2\pi f t)df \tag{42}$$

The frequency domain representation of a signal will consist of a set of peaks, and inspection of Equation 42 reveals that these peaks can roughly be interpreted as the relative amount that the frequency associated with said peak contributes to the original signal. The above Fourier transform is defined with respect to some signal $s \in \mathbb{R}^{\mathbb{R}}$. However, for most practical applications one only has access to a censored version of a signal defined over some interval $[a,b]$, i.e. $s_{[a,b]} \in \mathbb{R}^{[a,b]}$. For simplicity this interval will be considered to be $[0,T]$ giving rise to $s^{(T)} \equiv s_{[0,T]}$. One can get around this by rewriting $s^{(T)}(t) = s(t)\mathbb{I}_{[0,T]}(t)$. One can now use the convolution theorem, a well known theorem from Fourier theory, to evaluate the Fourier transform of the product of these functions, as the convolution of their Fourier transforms [32].

$$\tilde{s}^{(T)}(f) = F(s\cdot\mathbb{I}_{[a,b]})(f) = \Big(F(s) * F(\mathbb{I}_{[a,b]})\Big)(f)$$

The Fourier transform of the above indicator function is a complex phase shifted sinc function with width scaling inversely with T. This can be understood via the Fourier uncertainty principle, a signal's time localisation is inversely proportional to its frequency localisation.
In practice one does not have access to the continuous function $s^{(T)}(t)$, but rather a a set of N_s equispaced samples $(s_n)_{n\in\mathbb{N}_{0:N_s-1}}$, where $s_n = s^{(T)}(\frac{T}{N_s}n)$. One can thus form a numerical approximation of the fourier transform using a Riemann sum:

$$\hat{\tilde{s}}(f) = \frac{T}{N_s}\sum_{n=0}^{N_s-1} s_n \exp\left(-i2\pi f\frac{T}{N_s}n\right) = T\cdot\mathrm{DFT}(Tf)$$

Where DFT refers to the discrete Fourier transform, the reason for expressing this estimate in terms of the DFT is that the DFT has an efficient computational method for calculation

when N is a power of 2, this is called the Fast Fourier Transform (FFT) algorithm, it reduces computational complexity from $\mathcal{O}(N_s^2)$ to $\mathcal{O}(N_s \log(N_s))$. The DFT is typically restricted to evaluations on $k \in [0, N-1]$, which in turn restricts f to the Fourier modes $f = \frac{m}{T}$. The reason for this restriction is that it allows for DFT to be inverted [32].
It should be noted that $s^{(T)}$ is periodic, since $\hat{\tilde{s}}(f + 2mf_{\mathcal{N}}) = \hat{\tilde{s}}(f + 2m\left(\frac{N_s}{2T}\right)) = \hat{\tilde{s}}$, thus one need only calculate an interval of frequencies of length $2f_{\mathcal{N}}$ to capture its full behaviour. This is a consequence of digitising the signal and is called aliasing, frequencies higher than $f_{\mathcal{N}}$ have contributions that are indistinguishable from some $f \in [-f_{\mathcal{N}}, f_{\mathcal{N}}]$. Unless anti-aliasing is performed before ADC, whereby frequencies $f > f_{\mathcal{N}}$ are filtered out, then this leads to artefacts in the digitised signal, with certain frequencies in $[-f_{\mathcal{N}}, f_{\mathcal{N}}]$ being over represented due to higher frequency contributions. Additionally, for a real valued signal, if one is only interested in the Fourier transform amplitude $|s^{(T)}|$, one needs only calculate up to $[0, f_{\mathcal{N}}]$, since $s^{(T)}(-f) = \overline{s^{(T)}(f)}$. Figure 15 shows the approximated Fourier transforms of the waveforms displayed in Figure 14.

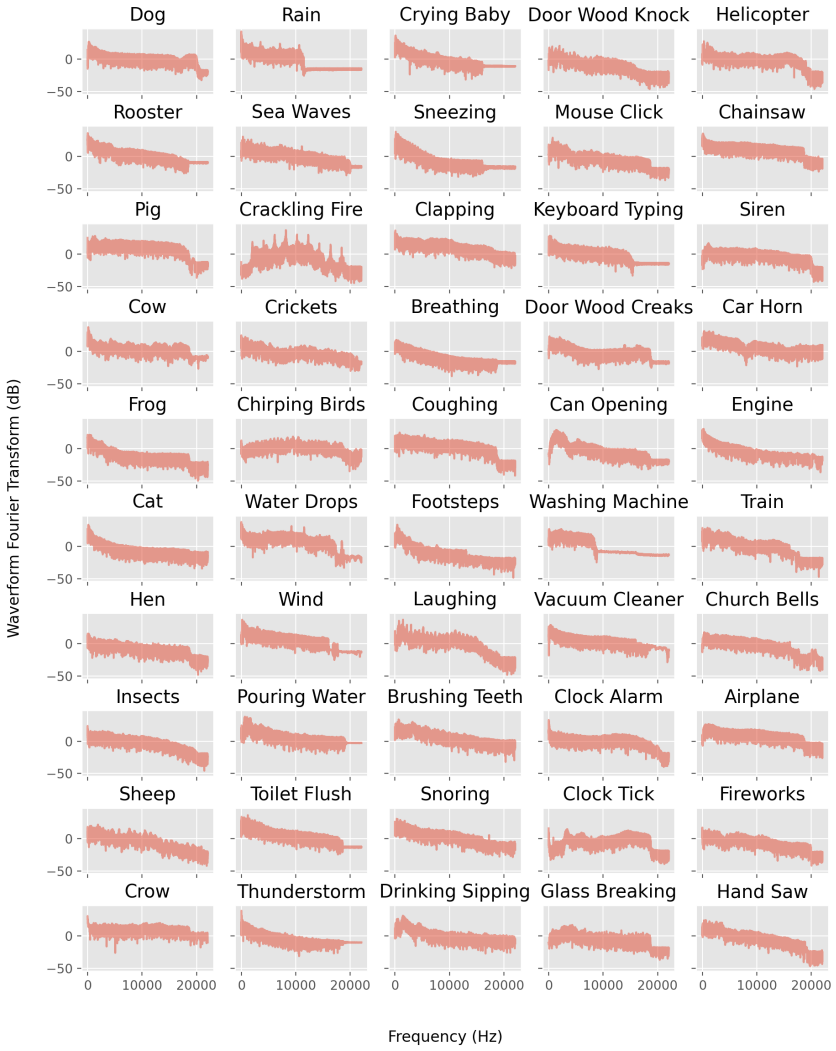

Figure 15: Magnitude of approximated Fourier transforms, using decibel scale, of the waveforms displayed in Figure 14.

3.2.2 The Short Time Fourier Transform (STFT)

Typically the types of signal one is dealing with are non-stationary, i.e. their frequency composition evolves with time. Applying the Fourier transform to such a signal will give a frequency representation akin to the time averaged amplitudes of the signal's frequency components. This is not very useful for the purposes of classification, since most spectrograms end up looking very similar, this phenomenon is observed in Figure 15.
A more suitable approach for non-stationary discrete signals is to take the discrete Short-Time Fourier Transform (STFT). This involves taking overlapping sub-sequences $S_m = (s_n)_{n\in\mathbb{N}_{(mh_s):(mh_s+N_w-1)}}$ of the signal, these sub-sequences have some specified length N_w and hop-siz h_s, which dictates the degree of overlap between adjacent sub-sequences. The idea behind using a STFT is for each window to be short enough such that frequency components time dependence is approximately constant. Typically, the shorter a signal is the louder it needs to be in order to be perceived, for typical signals this means that a windows length should correspond to at least 10, else the windows will be perceptually irrelevant. If this is to paired with a desire to exploit the computational efficiencies of the FFT, then for a sample frequency of $f_s = 44.1kHz$ sensible window sizes are $N_w = 512,\ 1024,\ 2048$.
As was mentioned previously, the Fourier transform of a censored signal can be seen as the convolution of the Fourier transform of the uncensored signal and the Fourier transform of the indicator function on the interval [0,T]. This corresponds with convolving the target Fourier transform with a complex phase shifted sinc function of width scaling inversely with $T_w = \frac{N_w}{f_s}$. Thus a small window size leads to a wide sinc function, which will cause what is referred to as spectral leakage in the Fourier transform, where a peaks energy (amplitude squared) leaks into its neighbouring frequencies. Furthermore, the functional form of the sinc function is a series of decaying lobes, which gives rise to a particularly undesirable form of spectral leakage called side-lobe leakage, this gives rise to fictitious peaks in the resulting Fourier transform [33]. The solution to this problem is to swap out the indicator function in for a more general window function w:

$$\tilde{s}^{(T)}(f) = F(s \cdot w(f) = (F(s) * F(w))(f)$$

one now has the freedom to select a windowing function that mitigates spectral leakage. This gives rise to the form of the STFT [32]:

$$\hat{\tilde{s}}_m(f;\ N_w,\ h_s,\ w) = \frac{T_w}{N_w} \sum_{n=0}^{N_w-1} s_{mh_s+n} w(n) \exp\left(-i2\pi f \frac{T_w}{N_w} n\right).$$

The time and frequency representations of three popular window functions are shown in Figure 16. One of these is the Hann window function, which by many is considered the default window function when performing STFTs, its bell shaped time representation gives rise to a similarly bell shaped frequency representation, with minimal side lobes. Its form is given by:

$$w_{\text{Hann}}(n) = \begin{cases} \sin\left(\pi \frac{n}{N}\right)^2, & t \in [0, N] \\ 0, & t \in \mathbb{R} \setminus [0, N] \end{cases} \text{ [33].}$$

This greatly reduces side lobe leakage, at the cost of a wider main lobe. However the resulting smearing effect this causes is considered better than the presence of artefacts caused by side lobe leakage.

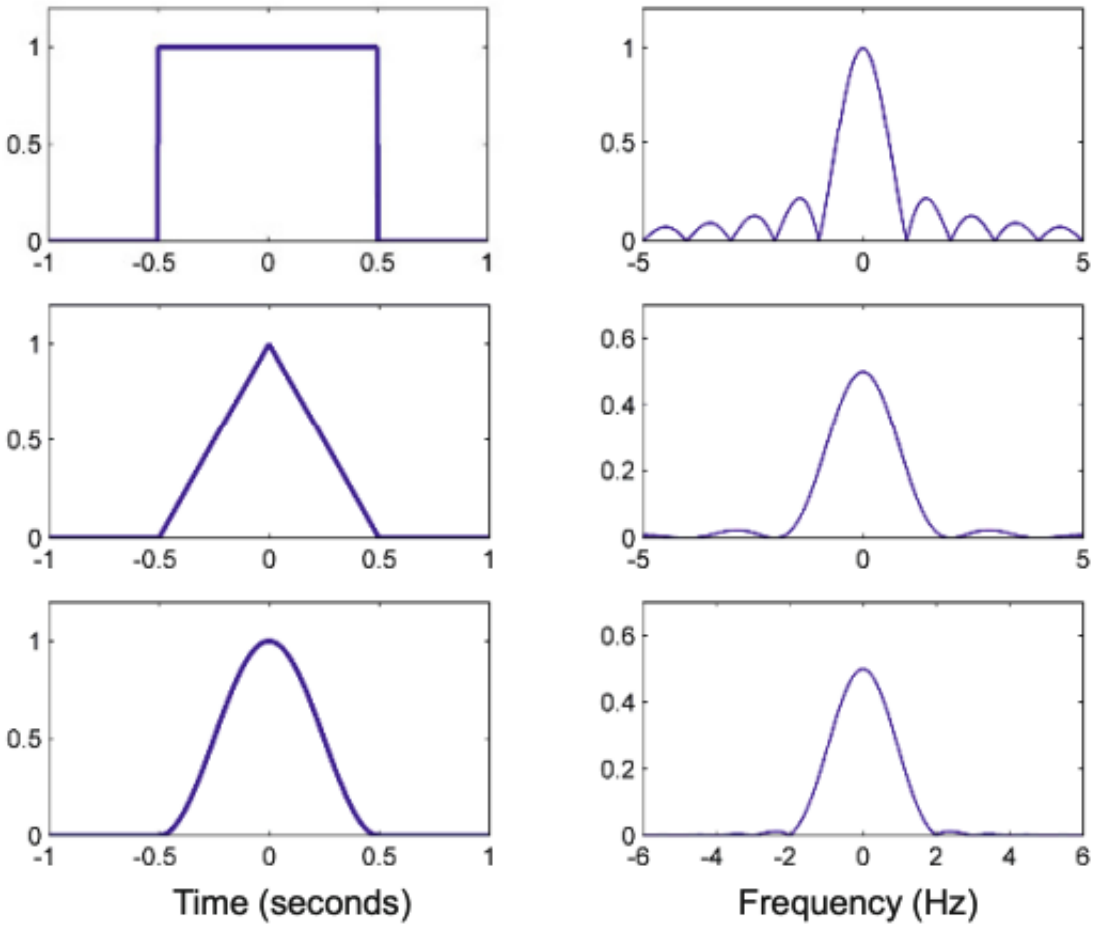

Figure 16: [32] A selection of window functions, using $(a, b) = (-\frac{1}{2}, \frac{1}{2})$, and their associated Fourier transform magnitudes. Shown from top to bottom is the indicator, triangular and Hann window functions.

A visualisation of a STFT being performed with the Hann window function is given in Figure 17. The most natural way to represent the STFT is to stack the squared magnitudes of the windowed Fourier transforms along a vertical axis, the horizontal axis can then be viewed as the window index. The squared magnitudes are used due to their connection to the energy contribution that said frequency component has to the wave, the energy being the quantity that determines the loudness of said frequency component. This stack of windowed Fourier transform squared magnitudes forms a matrix that can be visualised via a heat-map. This type of visualisation is called a spectrogram, it allows for one to view the frequency profile evolution of a signal. The Log-spectrograms for each example in the training data were calculated using the Python librosa library [34]. Log-spectrograms of the waveforms presented in Figure 14 can be seen in Figure. 18.

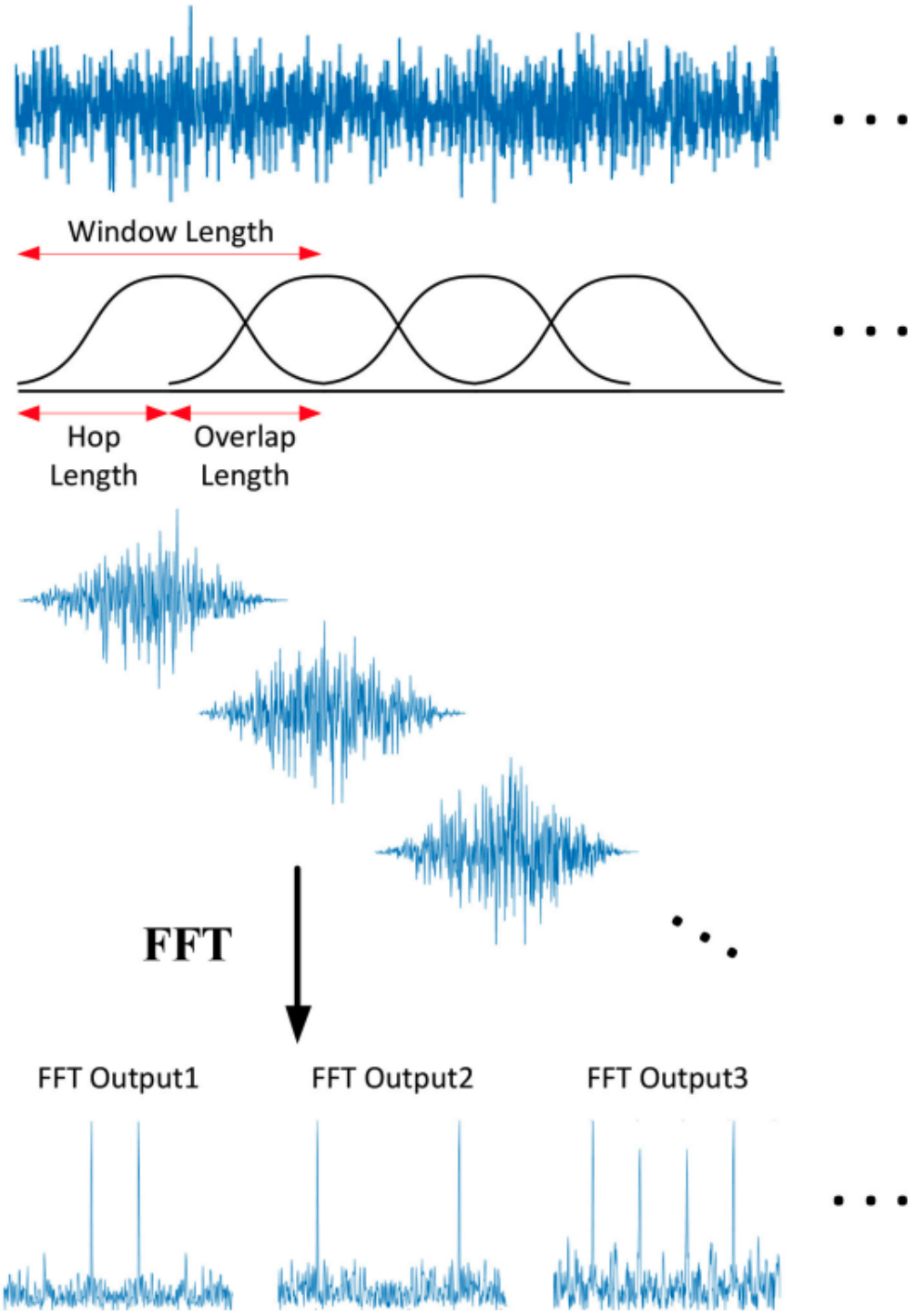

Figure 17: [35] A visualisation of the STFT procedure performed using a Hann window function.

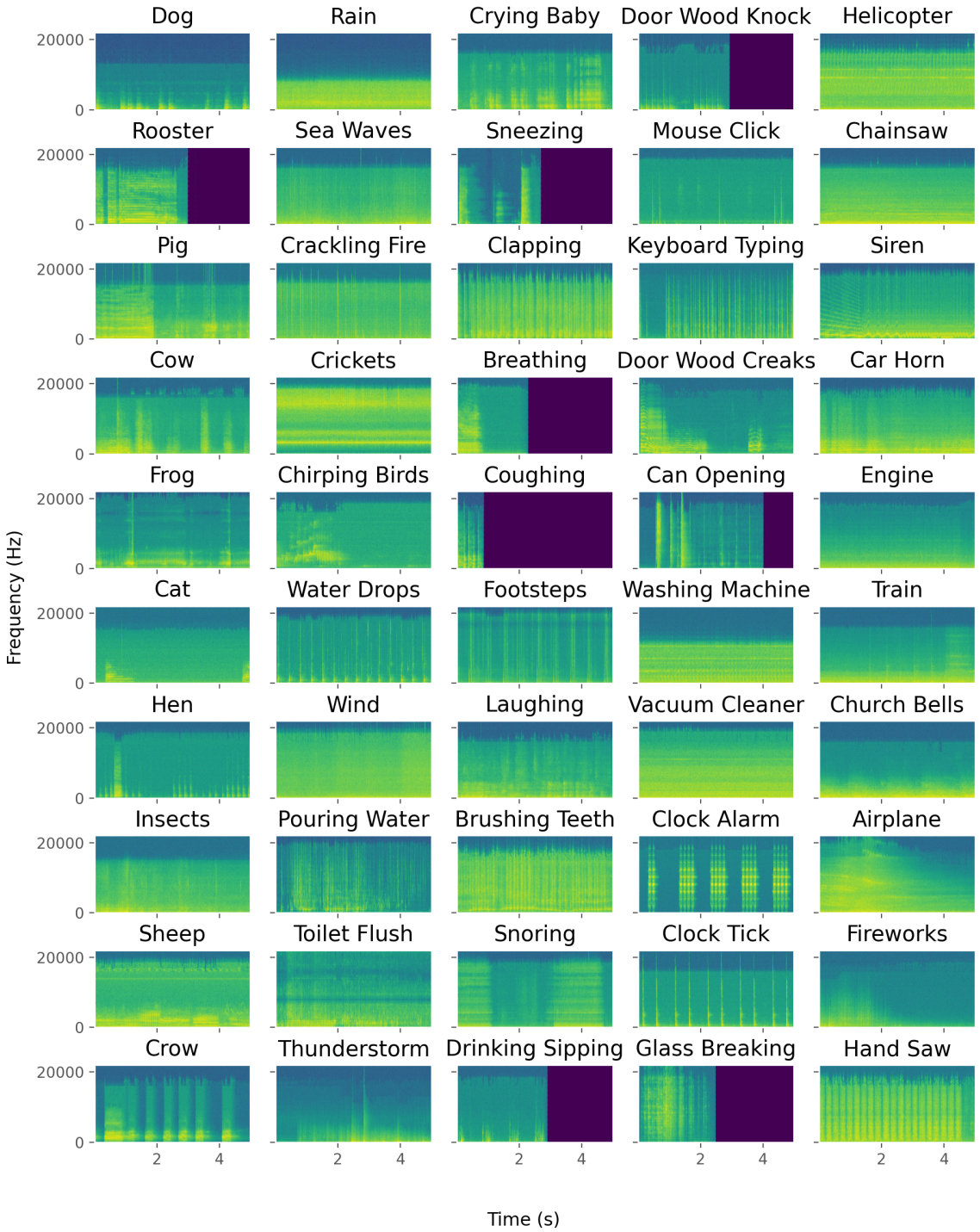

Figure 18: Log-spectrograms generated from the STFT($N_w = 1024,\ h_s = 512,\ w = w_{\text{Hann}},$) of the waveforms presented in Figure 14.

3.2.3 The Mel-Spectrogram

An inspection of Figure 10 shows that the log-spectrogram appears to be a more informative representation of the data than that of the raw waveform, shown in Figure 5, or the Fourier transform shown in Figure 6. However, a common trait amongst most of the spectrograms is the presence of complex structures appearing within the low frequency bands of the spectrogram, while high frequency bands of the spectrogram show little to no activity. It seems sub-optimal to be giving equal consideration to all frequencies present in the STFT, when lower frequency components appear to be more information dense. Furthermore, the spectrogram is quite a high dimensional representation, giving rise to an input space of dimension $N_w \times \left\lfloor \frac{N_s}{2h_s} \right\rfloor$. With these considerations in mind it would be worthwhile considering some form of dimensionality reduction that prioritises conserving low frequency information and compressing high frequency information.
Again, one can look towards humans for inspiration. Humans have evolved an efficient way of perceiving frequencies such that they can extract an information dense representation from the sound waves they receive. This representation being specifically adapted to environmental sounds that humans would encounter on a daily basis. A critical observation is that humans do not perceive frequencies on a linear scale. For example, a 100 Hz tone and a 200 Hz tone are easily distinguishable, whereas a 35100 Hz tone and a 35200 Hz tone will be indistinguishable to most, if not all, humans. It seems as though humans detect frequencies in bands, with the width of these bands appearing to grow as the frequency is increased, thus decreasing ones ability to resolve high frequencies. This is quantified mathematically by the Mel-scale, which is designed such that equally spaced frequencies on the Mel-scale sound equally perceptually spaced to the human ear [36]. The transformation between frequency and Mels has been empirically found to be given by Equation 43 [37].

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \tag{43}$$

The Mel-spectrogram is calculated from the spectrogram by applying n_M triangular filters to the frequency components constituting the original spectrograms. These n_M triangular filters are normalised and centred on equi-spaced frequencies on the Mel scale, with each filters starting point coinciding with the previous filters mid-point. The logarithmic nature of the Mel scale naturally gives rise to wider filters as frequency increases. An example of a 5 filter Mel filter bank is shown in Figure 19.

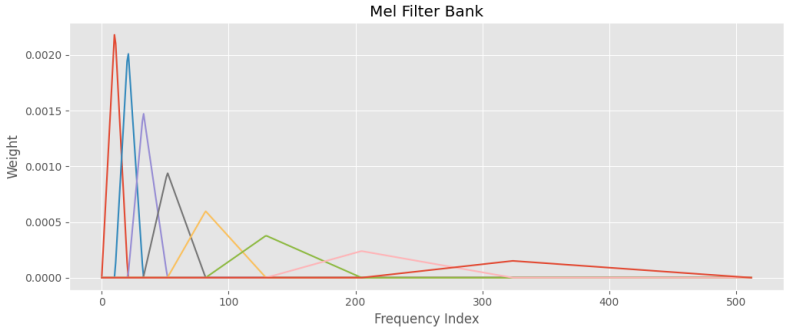

Figure 19: Mel filter bank containing 5 triangular filters.

Log-Mel-spectrograms for all examples in the dataset were calculated using the librosa library [34]. Additionally, Log-Mel-spectrograms of the waveforms presented in Figure 14 can be seen in Figure 20.

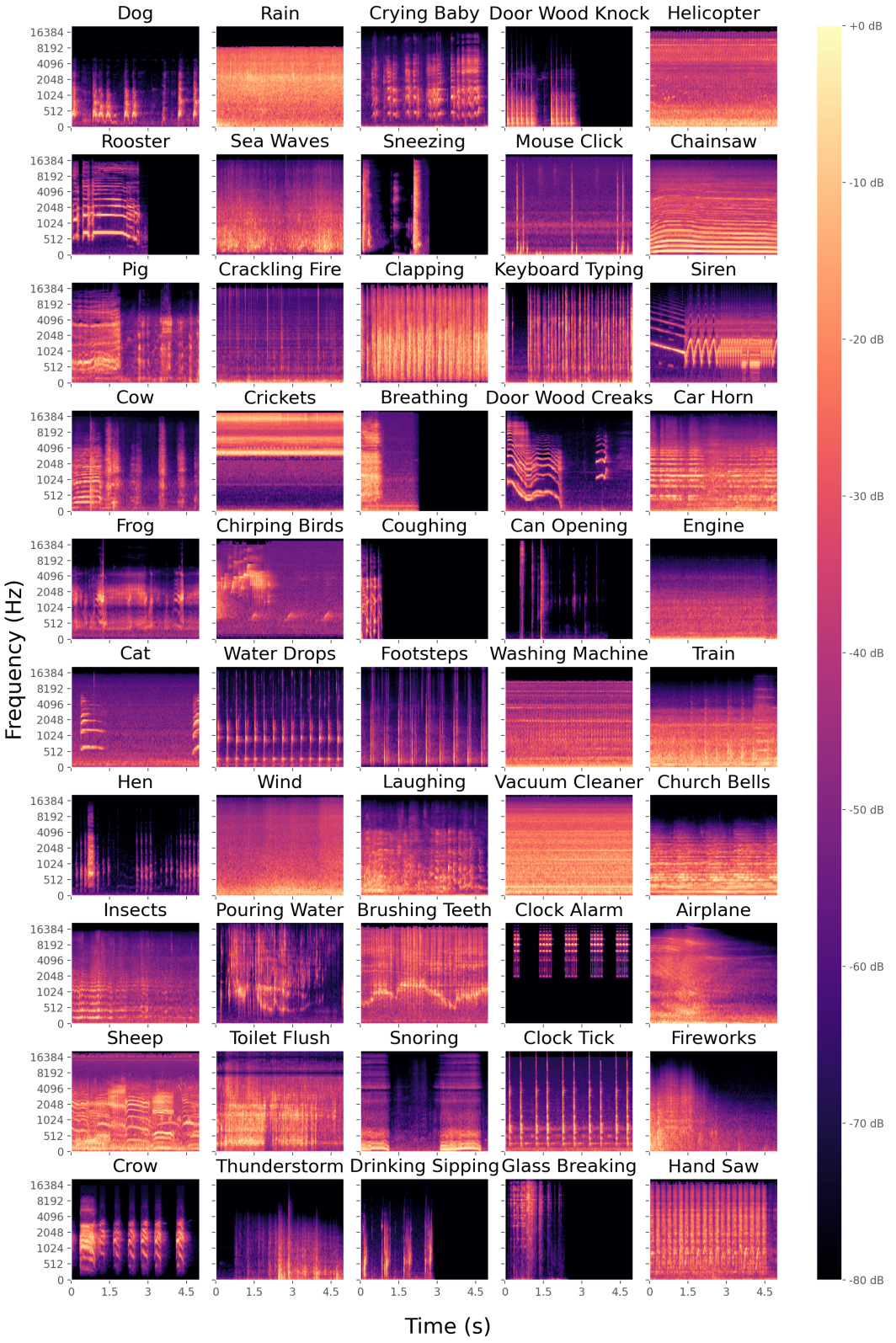

Figure 20: Log-Mel spectrograms of waveforms presented in Figure 14, the following parameters were used ($N_w = 1024,\ h_s = 512,\ w = w_{\text{Hann}},\ n_M = 128$).

3.3 Multi-Channel Log-Mel-Spectrogram Model

In order to achieve the objectives of this report a baseline networks performance needed to be determined, such that the performance of a comparable BNN could be compared. Kim (2020) reported state of the art performance on the ESC-50 dataset, reporting a 5-fold cross-validation classification accuracy of 89.50% [38]. However, the model in this paper was built for the "DCASE 2020 Task5 Urban Sound Tagging (UST)" dataset, this being a dataset of $T = 10$ s audio clips. This paper's experiments on the ESC-50 dataset were performed simply to demonstrate the networks performance on a small dataset. Additionally, the paper describes the usage of "Spatiotemporal context Pre-Processing" on "Spatiotemporal context" (STC) data, although it was not clear where exactly this came from. It does not seem to be relevant to the ESC-50 dataset and was likely some additional data that was provided with the UST dataset. The model architecture used in this paper is shown in Figure 21.

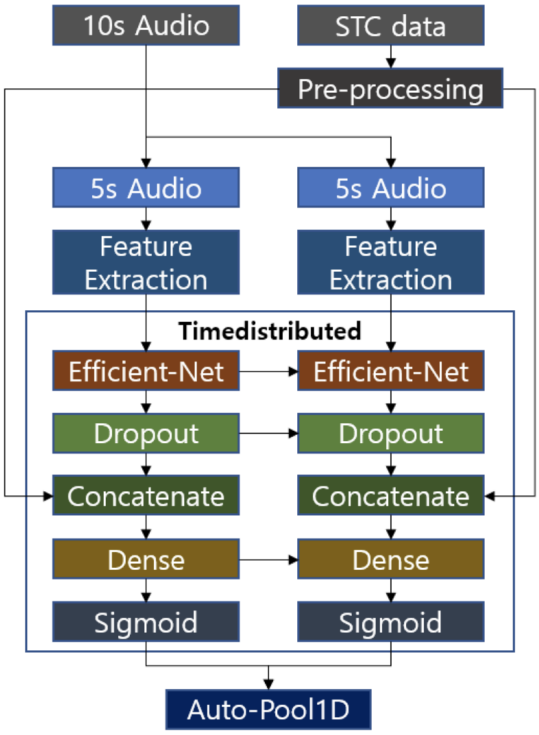

Figure 21: [38] Network architecture used in kim(2020) for audio classification on the USC dataset.

Unfortunately, it was not clear exactly how they applied this network intended for $T = 10$ s audio clips to the ESC-50 dataset which consists of $T = 5$ s audio clips. Additionally, it wasn't clear how they handled the lack of STC data associated with the ESC-50 dataset, when it appears the network architecture needs this STC data in order to operate. With these considerations in mind, there was insufficient information provided to replicate the ESC-50 result reported by this paper.
Nonetheless an attempt was made to try and replicate this model's success by taking inspiration from the reported architecture. The feature extraction step of the presented model pipeline involves taking what the paper describes as multi-channel log-Mel-spectrograms. These are calculated by first performing a median-filtering harmonic percussive source separation (HPSS) transformation to the raw waveforms. This

transformation can be seen as a method of separating the harmonic (temporally de-localised) and percussive (temporally localised) components of the waveform. The HPSS transform was performed using an implementation from the librosa library [34]. Next the log-Mel-spectrograms ($N_w = 1024,\ h_s = 512,\ w = w_{\text{Hann}},\ n_M = 128$) of the raw (R), harmonic (H) and percussive (P) components of the signal were calculated. These spectrograms are then stacked to form a 3-channel log-Mel-spectrogram [38]. A visualisation of the 3-channel log-Mel-spectrogram is shown in Figure 22.

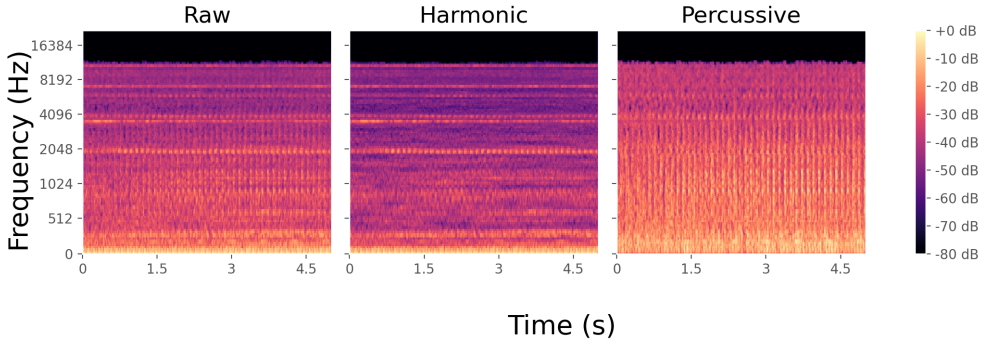

Figure 22: Visualisation of the 3-channels associated with a multi-channel log-Mel-spectrogram. The harmonic components can be seen as having a long temporal profile, and so appear as horizontal lines. Meanwhile the percussive components have a short temporal profile, which in accordance to the Fourier uncertainty principle gives rise to a wide frequency profile, hence their appearance as vertical lines.

These 3-channel log-Mel-spectrograms now have a similar form to a 3-channel RGB image. Interestingly, the paper exploits this similarity by performing transfer learning from an ImageNet trained network called EfficentNet. EfficentNet actually represents a family of architectures (B0-B7) of different sizes, these architectures are specifically designed to maximise the associated increase in performance per unit increase in required computation. They achieve this goal by using empirically derived ratios for how the networks input resolution, depth and width should scale relative to each other [39].
For experimentation purposes the pre-defined cross-validation folds associated with the ESC-50 dataset were ignored and the dataset was shuffled, else one runs the risk of overfitting to a specific fold. This is still sub-optimal since data from each test-fold is essentially still being used during experimentation to inform design choices. However, it is the best approach possible given the requirement that the final model needs to be trained and tested on the entire dataset via a 5-fold cross validation procedure, i.e. this problem is unavoidable.
An ANN was constructed using the Python TensorFlow library, this ANN took inspiration from the presented architecture in Figure 22. An EfficentNet(B4) was fed into a dropout layer ($p_d = 0.5$) which was then fed into a 50 node Softmax activation output layer. The original paper suggested an Adam($\eta = 0.0001$) optimizer, L2 regularisation with $\lambda = 0.0001$, a batch size of 16 and a number of training epochs equal to 10 [38]. The original paper did not mention wether the weights of the EfficentNet should be frozen, so both scenarios were investigated. In both scenarios the validation accuracy only reached $\approx 30\%$ while the training errors were $\approx 70\%$ and $\approx 90\%$ when the EfficientNet weights were frozen and-unfrozen respectively. A variety of different learning rates, regularisation hyperparameters and epoch number were experimented with to no avail, the severe overfitting problem persisted.

3.4 Data Augmentation

It seemed unusual that the work done by kim (2020) failed to mention the use of any data augmentation. After all the ESC-50 dataset only consists of 2000 examples. Nani et al. (2021) reported improvements in performance of $\approx 10\%$ when data augmentation was applied to this dataset, they also proposed a number of data augmentation algorithms to perform this task, in particular the Standard Signal Augmentation (SGN) algorithm. The (SGN) algorithm takes in a raw audio waveform and randomly applies a set of random augmentations to it, the algorithm is displayed in Algorithm 4 [40].

Algorithm 4 Standard Signal Augmentation

Input: Signal S
  Sample $A_1,\ A_2,\ A_3,\ A_4,\ A_5 \sim \text{Bernoulli}\left(\frac{1}{2}\right)$
  if $A_1 = 1$ **then**
    Speedup signal by a factor $R_S \sim \text{Uniform}(0.8,\ 1.2)$
  end if
  if $A_2 = 1$ **then**
    Shift pitch of signal by a number of semitones $R_P \sim \text{Uniform}(-2,\ 2)$
  end if
  if $A_3 = 1$ **then**
    Change volume of the signal by number of decibels $R_V \sim \text{Uniform}(-3,\ 3)$
  end if
  if $A_4 = 1$ **then**
    Inject white noise into signal with decibel signal to noise ratio $R_N \sim \text{Uniform}(0,\ 10)$
  end if
  if $A_5 = 1$ **then**
    Time shift signal by a number of seconds $R_T \sim \text{Uniform}(-0.005,\ 0.005)$
  end if

Nani et al (2021) used this algorithm via an in-built MATLAB implementation, no such implementation existed in Python and as such one was written. This implementation was then used to increase the size of the dataset by a factor of 10. This data augmentation process is visualised in Figure 23.

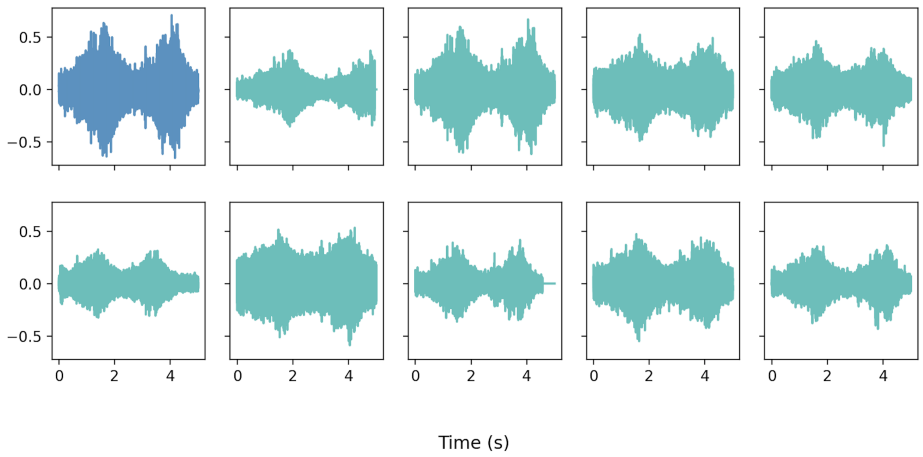

Figure 23: A non-augmented waveform alongside 9 augmented waveforms generated from it via the SGN algorithm.

3.5 Raw Waveform Model

Discouraged by the poor performance of the previously investigated architecture a different approach was investigated. Rather than using log-Mel-spectrograms, a high performing network using raw audio waveforms was searched for. One such family of models was the ACDNet family of architectures presented by Mohaimenuzzaman et al. (2021), which reports a 5-fold cross-validated classification accuracy on ESC-50 of 87.1% [41]. The family of architectures is parameterised by the signal length, which is referred to as i_{len} in this paper. A diagram of the ACDNet family of architectures is shown in Figure 24.

Layers	Kernel Size	Stride	Filters	Output Shape	Block
Input				$(1,\ 1,\ w = i\_len)$	
conv1	$(1,\ 9)$	$(1,\ 2)$	8	$(8,\ 1,\ w = \frac{w-9}{2} + 1)$	
conv2	$(1,\ 5)$	$(1,\ 2)$	64	$(64,\ 1,\ w = \frac{w-5}{2} + 1)$	SFEB
Maxpool1	$(1,\ ps)$	$(1,\ ps)$		$(64,\ 1,\ w = \frac{w}{ps})$	
swapaxes				$(1,\ 64,\ w)$	
conv3	$(3,\ 3)$	$(1,1)$	32	$(32,\ 64,\ w)$	
Maxpool2	$(2,\ 2)$	$(2,\ 2)$		$(32,\ 32,\ w = \frac{w}{2})$	
conv4,5	$(3,\ 3)$	$(1,\ 1)$	64	$(64,\ 32,\ w)$	
Maxpool3	$(2,\ 2)$	$(2,\ 2)$		$(64,\ 16,\ w = \frac{w}{2})$	
conv6,7	$(3,\ 3)$	$(1,\ 1)$	128	$(128,\ 16,\ w)$	
Maxpool4	$(2,\ 2)$	$(2,\ 2)$		$(128,\ 8,\ w = \frac{w}{2})$	
conv8,9	$(3,\ 3)$	$(1,\ 1)$	256	$(256,\ 8,\ w)$	
Maxpool5	$(2,\ 2)$	$(2,\ 2)$		$(256,\ 4,\ w = \frac{w}{2})$	TFEB
conv10,11	$(3,\ 3)$	$(1,\ 1)$	512	$(512,\ 4,\ w)$	
Maxpool6	$(2,\ 2)$	$(2,\ 2)$		$(512,\ 2,\ w = \frac{w}{2})$	
Dropout (0.2)					
conv12	$(1,\ 1)$	$(1,\ 1)$	n_cls	$(n\_cls,\ 2,\ 4)$	
Avgpool1	$(2,\ 4)$	$(2,\ 4)$		$(n\_cls,\ 1,\ 1)$	
Flatten				(n_cls)	
Dense1			n_cls	(n_cls)	
Softmax				(n_cls)	Output

Figure 24: [41] Diagram of the ACDNet family of architectures, with i_{len} corresponding to the length of the signal entering the network and $ps = \frac{wf_s}{100i_{len}}$. Each hidden layer uses a ReLU activation function and is followed by a Batch normalization layer. Note, this paper uses a $(C,\ H,\ W)$ convention for listing input/output shapes.

The ACDNet architecture can be broken down into 3 blocks, the Spectral Feature Extraction Block (SFEB), the Temporal Feature Extraction Block (TFEB) and the output block. The SFEB applies two 1-dimensional convolutional layers followed by a max pooling layer. This max pooling layer uses a large pool size, for $i_{len} = 220500$ the pool size is $ps = 110$.

Combining this with the stride size of 2 present in the previous two layers and one can expect a large reduction in dimensionality when comparing the input representation to the representation entering the TFEB. This is an intelligent design choice as it will add a significant amount of translation invariance to the CNN while simultaneously reducing model complexity. Translation invariance is a useful property in this context since translating a signal has no bearing on its class. Before this intermediate representation enters the channel and height dimension of the representation are swapped. This causes the convolutions performed in the TFEB to occur over botht the signals time and frequency domain [41].
The above ACDNet architecture was implemented using Python's TensorFlow library. This ACDNet was trained on the SGN augmented dataset, it should be noted that the original ACDNet paper used a different type of augmentation. The original ACDNet paper reported training for 2000 epochs using a scheduled learning rate and a Nesterov momentum ($\alpha = 0.9$) SGD optimizer. if i denotes the current training epoch then this learning rate schedule can be summarised as follows:

$$\eta(i) = \begin{cases} 0.01, & 1 \leq i \leq 10 \\ 0.1, & 10 < i \leq 600 \\ 0.01, & 600 < i \leq 1200 \\ 0.001, & 1800 < i \end{cases}.$$

Additionally, the paper reported using L2 regularisation ($\lambda = 5 \times 10^{-4}$) and a batch size of 64.
Significant improvements in performance were observed when using the approach outlined above when compared to the previous multi-channel log-Mel-spectrogram model. The validation accuracy reached $\approx 50\%$, however there was still a significant overfitting problem, with the training accuracy reaching $\approx 95\%$. In an attempt to get closer to the original papers reported accuracy, its procedure was followed more closely. The original paper dealt with a input signal length of $i_{len} = 30225$. This dimensionality reduction was achieved by down-sampling the signal to 20kHz, the justification for this being that the authors deemed frequency contributions above $f_N = 10kHz$ as insignificant. The remaining dimensionality reduction was achieved by extracting ≈ 1.51 s windows from each audio waveform and using this as the input [41]. Unfortunately after following this updated procedure the generalisation gap was only slightly improved, with validation accuracy $\approx 55\%$ and testing accuracy $\approx 95\%$. An attempt was made to close the generalisation gap by reducing the complexity of the model. This involved removing the convolutional blocks of the 256 and 512 filter maps from the TFEB, then adjusting the average pooling layer's pool size to account for the larger grid it was now being applied to. This effort was unsuccessful.

3.6 Data Augmentation Revisited

The difference between the performance reported by Mohaimenuzzaman et al. (2021) for the ACDNet architecture and the performance achieved by the ACDNet implementation used in this paper was very suspicious. The only difference between these approaches was their associated data augmentation procedures, yet Nani etal. (2021) had reported SGN to be an effective data augmentation procedure [40]. To investigate this further a script was written to first augment waveforms via SGN and then export them to .wav files. Upon listening to the augmented audio it was clear something was wrong, mainly the range of signal to noise ratios allowed by SGN was far too low. Any augmented sample which had

the additive white noise augmentation could very easily be identified as having arisen from the data augmentation procedure. This being a significant problem due to approximately half of the augmented examples having this augmentation applied. As was discussed earlier in the Literature Review, the data augmentation process needs to mimic the actual data generation process for it be useful. The ESC-50 dataset is not very noisy, thus models trained on data from the data augmentation process will go through covariate shift when evaluated on testing data, this explains the observed poor generalisation. The above considerations imply that the model in presented in Nani etal. (2021) performs well in spite of the additive white noise augmentation rather than because of it.
A number of different ranges for the white noise decibel signal to noise ratio modifier were experimented with, however all of them gave rise to augmented samples that were immediately distinguishable as being augmented. Thus, it was decided to drop the additive white noise augmentation entirely. Additionally the allowed random time shifts were given a much larger range of $[-0.25, 0.25]$ and the decibel volume gain was also given a wider range of values $[-5, 5]$. This Adjusted Standard Signal Augmentation (ASGN) algorithm is presented in Algorithm 5.

Algorithm 5 Adjusted Standard Signal Augmentation

Input: Signal S
  Sample $A_1,\ A_2,\ A_3,\ A_4 \sim \text{Bernoulli}\left(\frac{1}{2}\right)$
  if $A_1 = 1$ **then**
    Speedup signal by a factor $R_S \sim \text{Uniform}(0.8,\ 1.2)$
  end if
  if $A_2 = 1$ **then**
    Shift pitch of signal by a number of semitones $R_P \sim \text{Uniform}(-2,\ 2)$
  end if
  if $A_3 = 1$ **then**
    Change volume of the signal by number of decibels $R_V \sim \text{Uniform}(-5,\ 5)$
  end if
  if $A_3 = 1$ **then**
    Time shift signal by a number of seconds $R_T \sim \text{Uniform}(-0.25,\ 0.25)$
  end if

To avoid a repeat of the issues associated with the previous augmentation procedure a hypothesis test was conducted on 100 audio samples. With probability $\frac{1}{2}$ these samples either had ASGN applied to them or they were left alone, the results of these applications were stored in an unseen text file. These audio samples were listened to and classified as either being augmented via ASGN or non-augmented. Under the null hypothesis that ASGN produces signals that are indistinguishable by human hearing to non-augmented signals then the number of correct classifications C is binomially distributed $C \sim \text{Binomial}\left(100, \frac{1}{2}\right)$. Out of the 100 presented samples $C = 56$ had their origin correctly identified, the one-sided p-value associated with this test statistic is 0.1356, thus, the null hypothesis fails to be rejected. This validating the usage of the ASGN for data augmentation.

3.7 Segmented Log-Mel-Spectrogram

Unfortunately, the poor data augmentation procedure had cost a lot of time at this point, since training the ACDNets was a very time consuming process. This motivated a shift in the project's direction, away from investigating state of the art models, which often require days of time to train over all 5 training folds, and more towards finding a basic baseline model such that the objectives of this investigation could be fulfilled. The default baseline model for ESC-50 trained ANNs is given by Piczak (2015), which reports a 5-fold cross-validated classification accuracy of 64.50%. This paper describes a very simple CNN, consisting of 2 convolutional layers and 2 fully connected layers, its architecture is shown in Figure 25 [42].

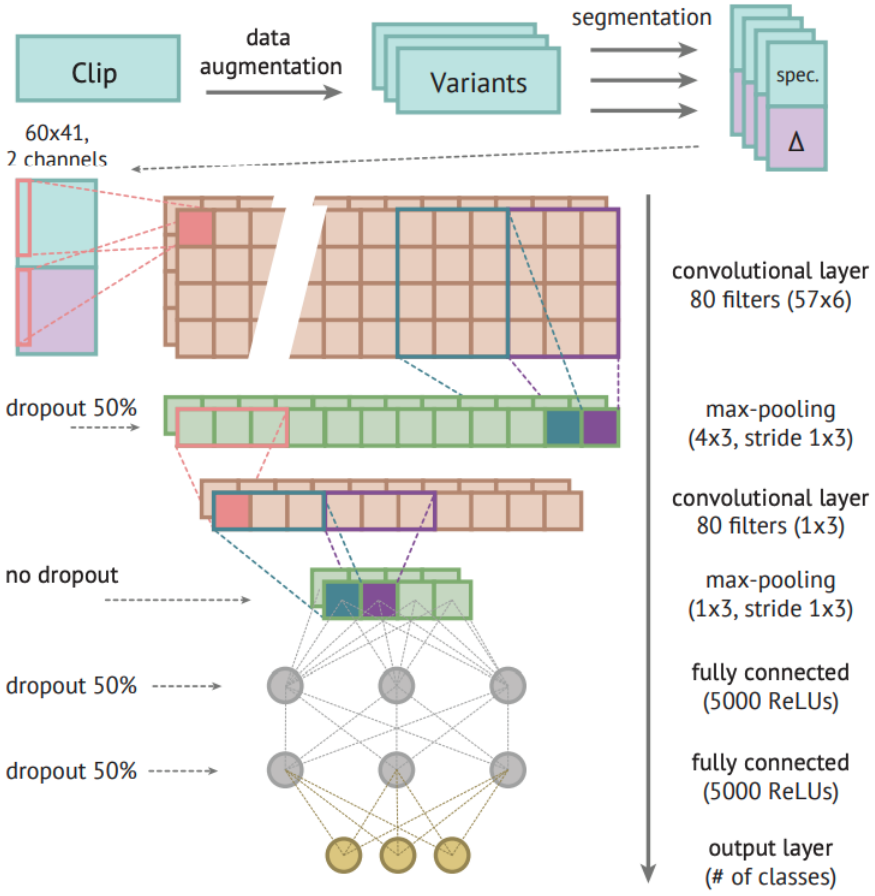

Figure 25: [42] Network architecture used for ESC-50 classification by Piczak (2015).

An inspection of Figure 25 reveals some initial short comings of this network architecture. The fully-conected layers both contain 5000 nodes, this is a very large number when one considers that the weight matrix connecting these two layers belongs to $W^{(4)} \in \mathbb{R}^{5000\times5000}$, i.e. has 25 M components. This did not bode well for the new focus of finding a baseline model that can be quickly trained. It was also believed that this is a wasteful usage of parameters. For this reason a novel architecture was developed.
Although, the baseline model in Piczak (2015) will not be used , this paper still presents a very interesting method of pre-processing. The first stage of pre-processing is augmentation of the raw waveforms, the augmentation performed in this paper is not too dissimilar to that described by ASGN. The augmented waveforms then go through segmentation,

whereby windows of audio samples are taken, with some specified window N_w length and hop h_s size. The benefits of segmentation are two fold, the number of training examples is increased and these training examples have had their dimensionality reduced. Finally the log-Mel-spectrograms and local estimates of their derivatives are calculated, the log-Mel-spectrograms are then stacked on-top of their associated local derivative estimate to give a 2-channel input.
Once it comes to testing the model, all of the segments associated with a given example have their predictions calculated by the model, the prediction for the example are then given by aggregating the predictions over its segments. Piczak (2015) showed that a probability voting scheme, i.e. taking the mean Softmax outputs over segment predictions, was better than performing a majority voting scheme, i.e. taking the modal prediction over segmentation predictions. Hence the latter will be performed.
A slightly modified pre-processing method to that of Piczak (2015) was performed. First log-Mel-spectrograms ($N_w = 1024,\ h_s = 512,\ w = w_{\text{Hann}},\ n_m = 128$) were calculated. Then local estimates of these log-Mel-spectrogram's derivatives were calculated and stacked on top of one another to produce a 2 channel features. Next windowing was performed with $N_w = 128$ and $h_s = 64$. Each window corresponding to approximately 1.5 s of audio data, this roughly coinciding to the length scale that Mohaimenuzzaman et al. (2021) determined to be effective [41].
The neural network designed to tackle this segmented audio classification problem was a CNN, its architecture is shown in Figure 26. The network has a depth D=6, with 4 convolutional hidden layers and 2 fully connected layers. The depth of this ANN would be considered slightly above the deep learning threshold, this motivates the usage of batch normalisation layers after each layer convolutional / fully-connected layers in order to mitigate internal covariate shift. The convolutional layers at the start of the network have large kernel sizes, this is to allow for the effective receptive field of the deeper convolutional layers to essentially cover the entire input, and thus be able to detect more complex phenomenon. Additionally the first pooling layer has a slightly larger pool size compared to others, this is to maximise the translation invariance properties of the network, this is especially relevant at the start of the network since background knowledge dictates that the translation of a waveform does not provide discriminating information about its class. The later pooling layers perform average pooling, since max pooling is best suited to sparse representations and it is thought that the later intermediate representations would not be as sparse, thus it would be better trying to take contributions from all nodes being aggregated over. The output layer uses a Softmax activation function whilst the hidden layers use ELU activation functions, this choice was made to mitigate the dying node problem associated with ReLU activation functions. Regularisation has been applied using L2 regularisation with $\lambda = 10^{-4}$. Additional regularisation is supplied via the presence of dropout layers after each hidden layer, the dropout probabilities of which are $p_d = 0.2$ if the next layer is convolutional and $p_d = 0.5$ if it is full connected.
Figure 26 also displays a comparable BNN for the presented CNN. The main difference being that the output layer is now Bayesian, with a prior over its learnable parameters equal to an isotropic Gaussian. Additionally the BNN has had its L2 regularisation removed as it was found to be causing the model to under-fit. Although this isn't a full Bayesian neural network it will hopefully enjoy some of the benefits associated with BMA approximation. Additionally each of these networks was used to generate a deep ensemble of 5 trained networks, this was to further the investigation into Bayesian model averaging. There is also a difference in the learning rates used for these different models, both used an RMSProp optimizer, however the CNN used $\eta = 0.001$ and the BNN used $\eta = 0.01$. During

training, a single sample was used to generate the Monte Carlo estimates of the loss and its associated gradient, this has been shown to be effective in the literature [15]. For testing, the VPD was calculated by using a 10 sample Monte Carlo estimate from the variational approximation of the weight posterior.

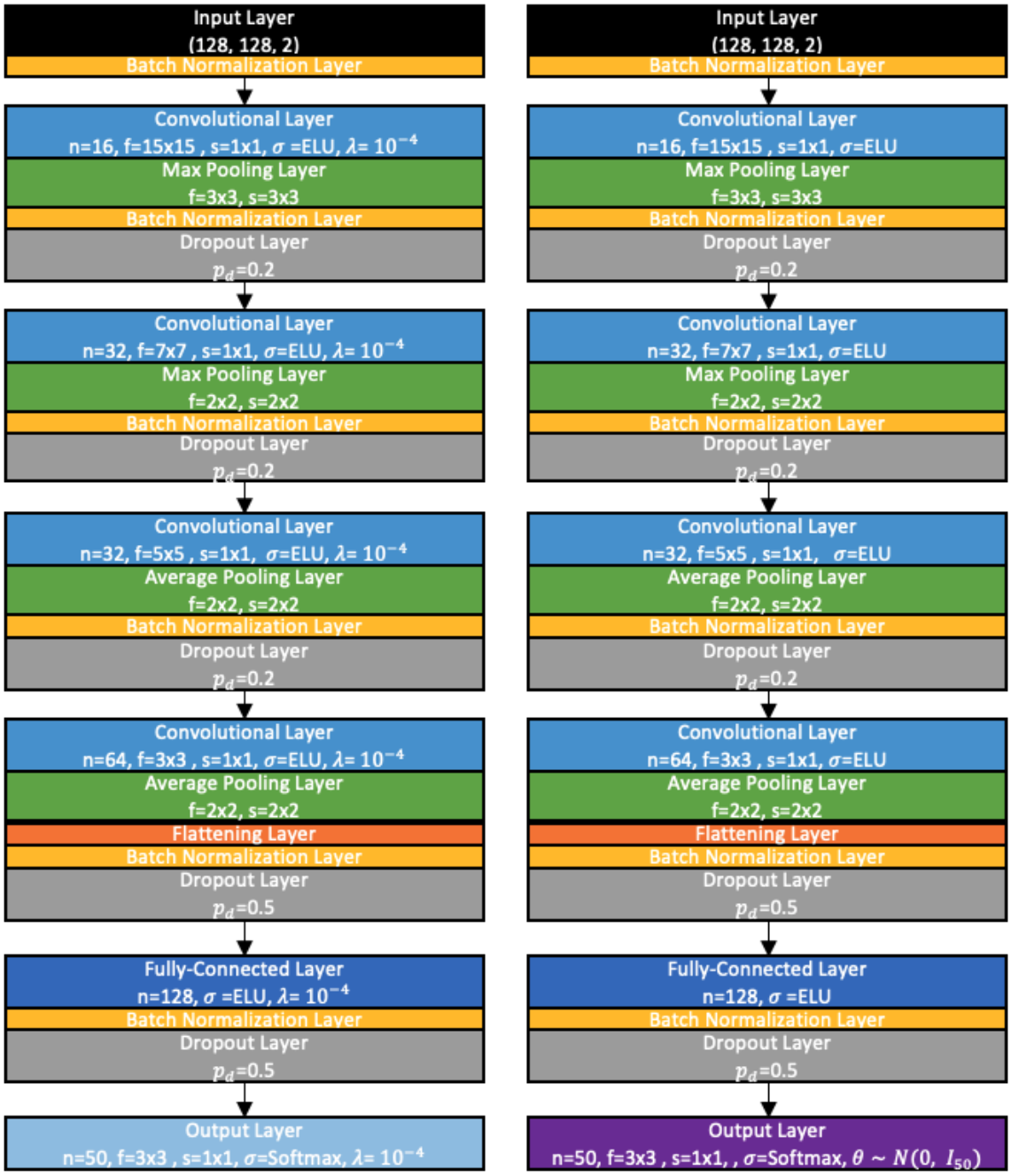

Figure 26: Diagram of baseline CNN architecture (left) and a comparable BNN architecture (right).

4 Results

4.1 Learning Curves

4.1.1 CNN

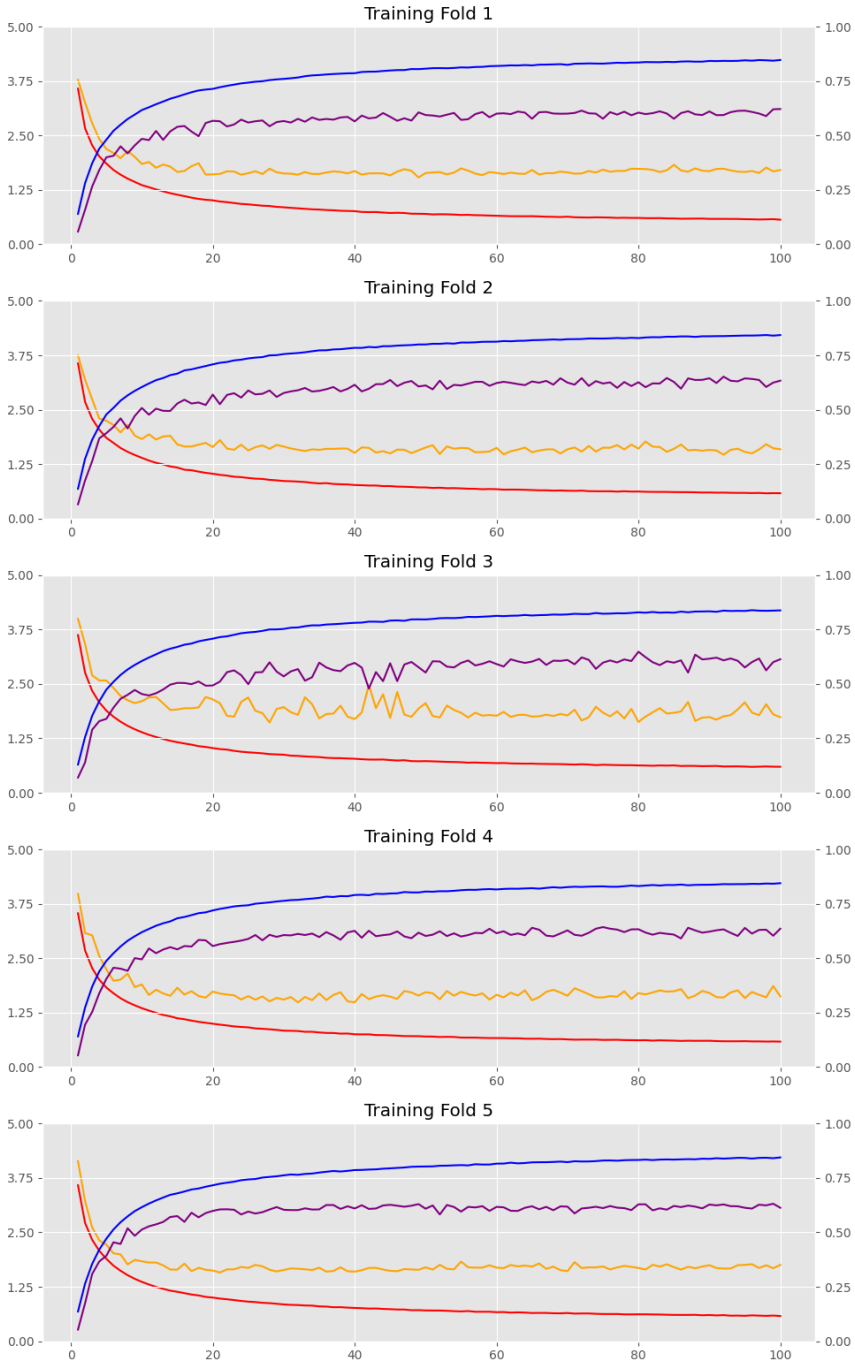

Figure 27: Learning curve for CNN described in Figure 26. The blue, purple, orange and red curves represent the test accuracy, validation accuracy, validation loss and test loss respectively.

4.1.2 BNN

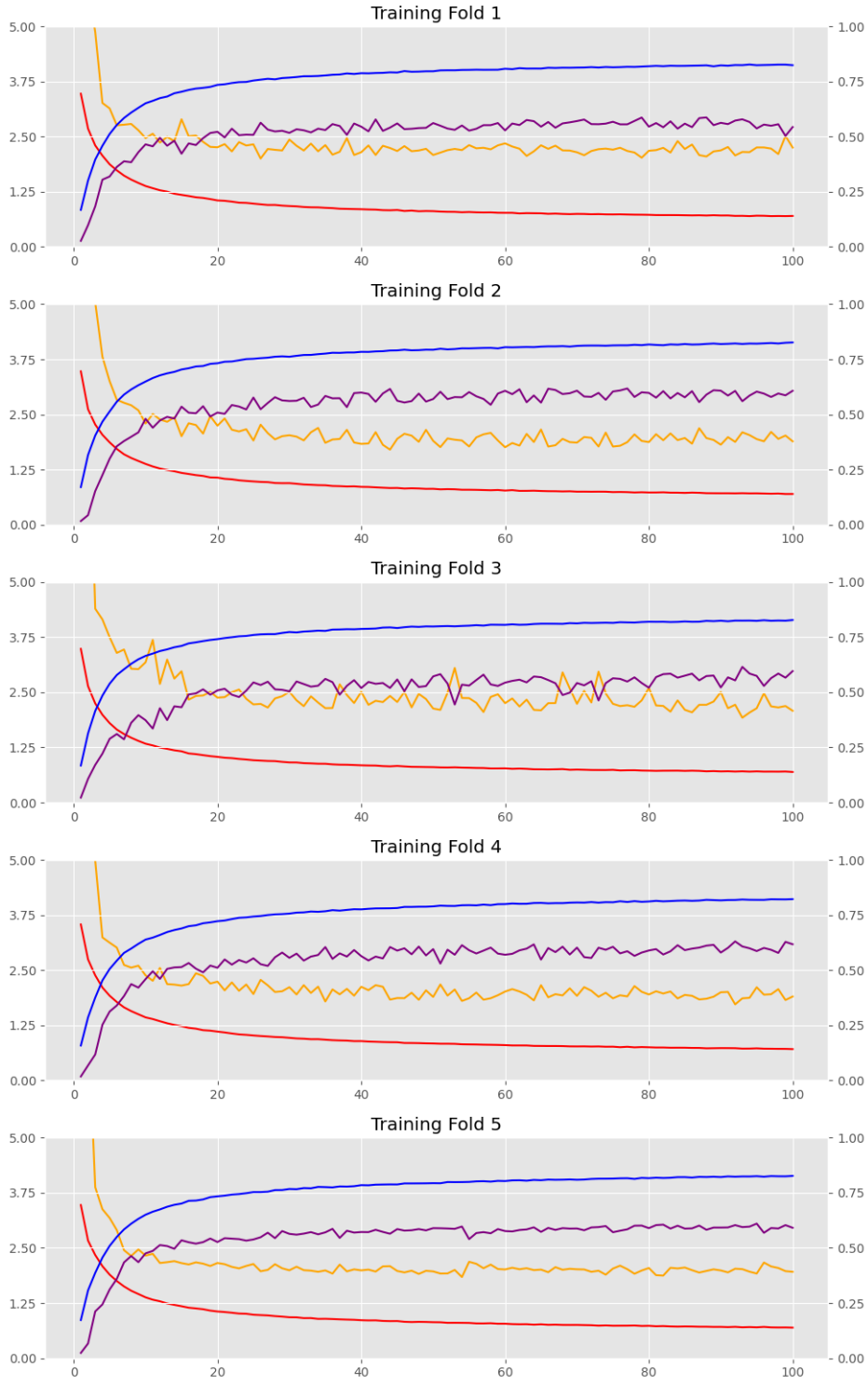

Figure 28: Learning curve for BNN described in Figure 26. The blue, purple, orange and red curves represent the test accuracy, validation accuracy, validation loss and test loss respectively.

4.1.3 CNN Ensemble

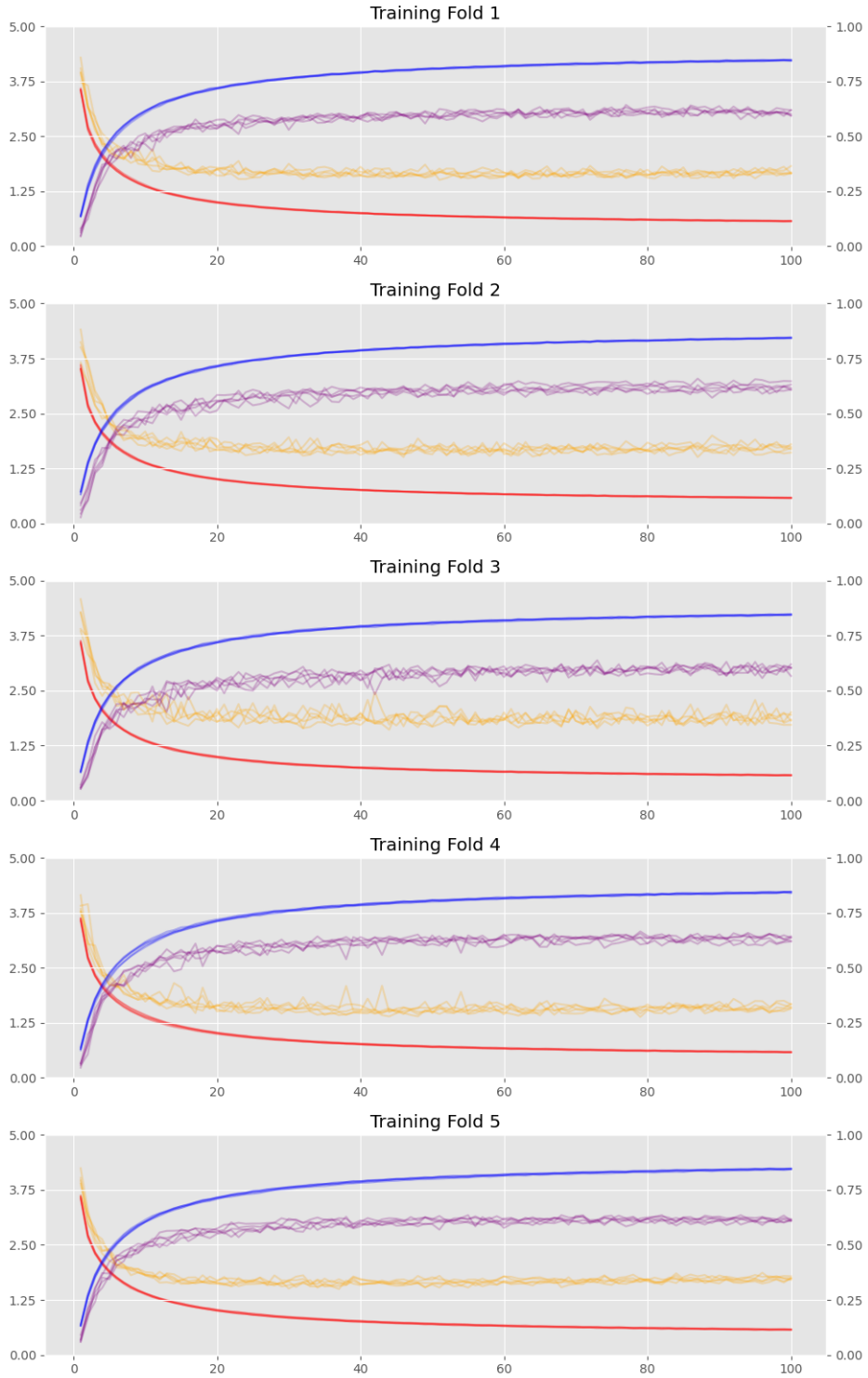

Figure 29: Learning curve for ensemble of CNNs described in Figure 26. The blue, purple, orange and red curves represent the test accuracy, validation accuracy, validation loss and test loss respectively.

4.1.4 BNN Ensemble

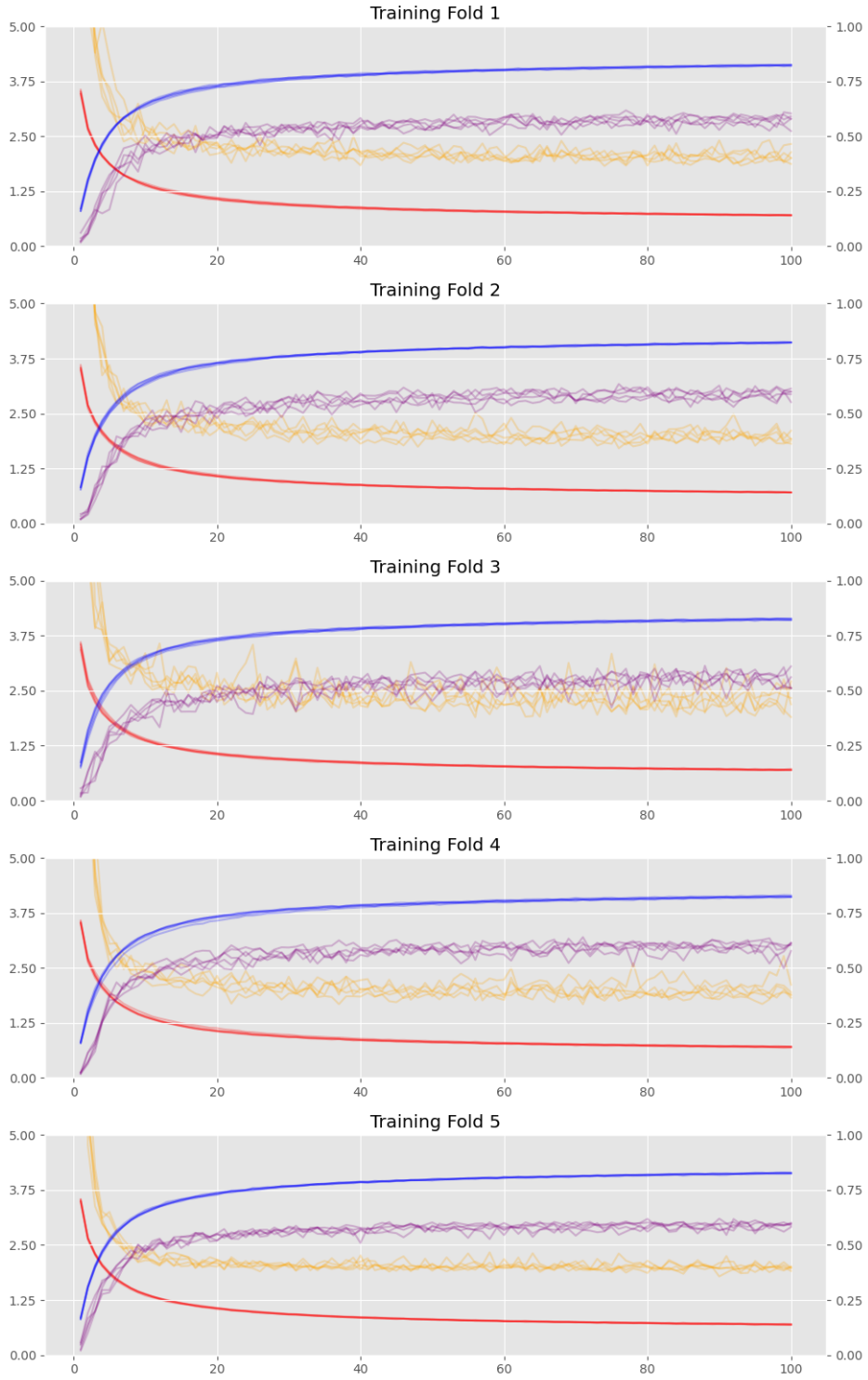

Figure 30: Learning curve for ensemble of BNNs described in Figure 26. The blue, purple, orange and red curves represent the test accuracy, validation accuracy, validation loss and test loss respectively.

4.2 Confusion Matrices

4.2.1 CNN

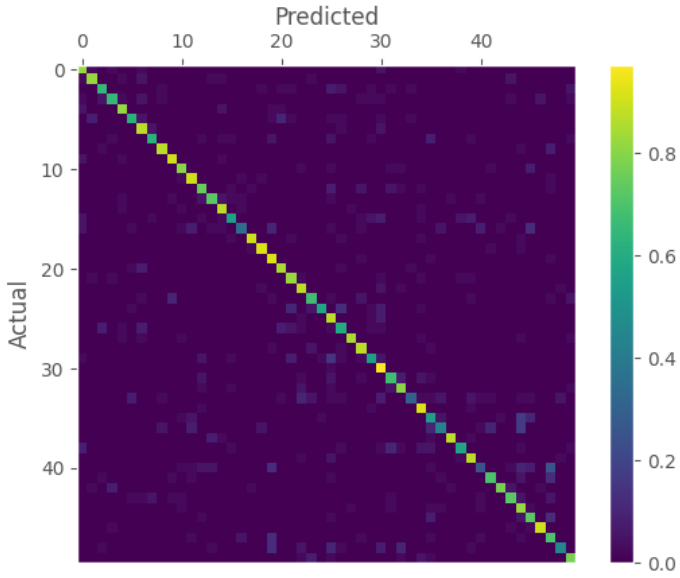

Figure 31: Confusion matrix of CNN model.

4.2.2 BNN

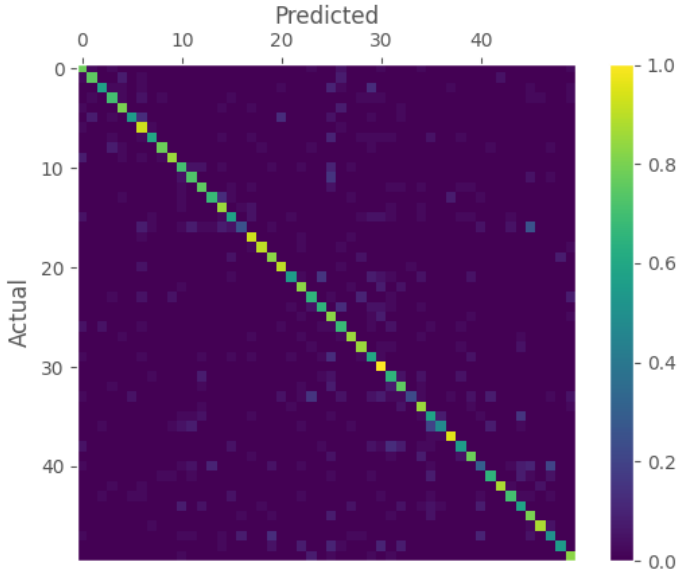

Figure 32: Confusion matrix of BNN model.

4.2.3 CNN Ensemble

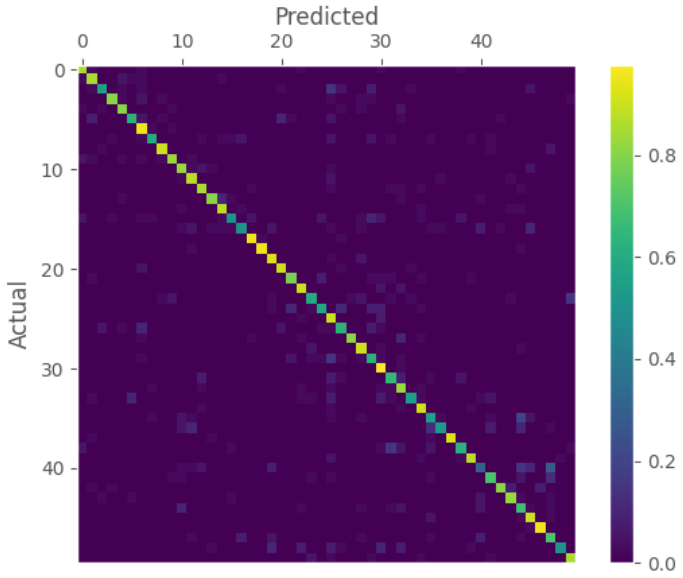

Figure 33: Confusion matrix of CNN ensemble model.

4.2.4 BNN Ensemble

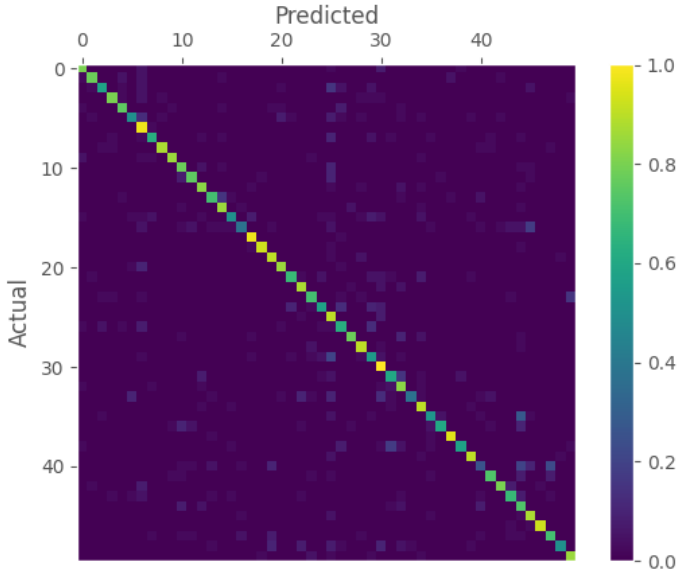

Figure 34: Confusion matrix of BNN ensemble model.

4.3 5-fold Cross Validation Accuracies

Model	$\hat{A}_{\text{val}}^{(1)}$	$\hat{A}_{\text{val}}^{(2)}$	$\hat{A}_{\text{val}}^{(3)}$	$\hat{A}_{\text{val}}^{(4)}$	$\hat{A}_{\text{val}}^{(5)}$	$\hat{A}_{\text{cv}}$	$\hat{\sigma}_{\text{cv}}$
CNN	0.7040	0.7550	0.7275	0.7264	0.7125	0.7251	0.0078
BNN	0.6575	0.7050	0.7250	0.7475	0.6950	0.7060	0.0135
CNN Ensemble	0.7525	0.7600	0.7575	0.7825	0.7475	0.7600	0.0054
BNN Ensemble	0.7225	0.7300	0.7050	0.7725	0.7500	0.7360	0.0104

Table 3: Summary of the 5-fold cross validation procedure performed over the CNN, BNN, CNN Ensemble and BNN Ensemble Models.

5 Discussion

The learning curves displayed in Figures 27, 28, 29 and 29 show that all the models investigated appear to have converged and training was not stopped prematurely. Furthermore the learning curves of the CNN models displayed in Figures 27 and 29 appear to be far more stable than those shown in Figure 28 and 30. This could potentially be due to the higher learning rate associated with the BNN's optimisation, although it is likely a result of the Monte-Carlo estimation of the loss and gradients. This instability also manifests in the cross validation estiamtes displayed in Table 3. Interestingly, Table 3 shows much higher performances than those displayed in the validation accuracies displayed in Figures 27, 28, 29 and 29, this is due to the probability voting scheme used when aggregating the segmented predictions associated with a given example during testing. There do not appear to be any noticeable trends across the confusion matrices of the investigated models, these being shown in Figures 31, 32, 33 and 34. This is somewhat surprising, one may have assumed there would be a 10×10 block like structure to the confusion matrices arising from the 5 class categories associated with this dataset.
The results shown in Table 3 reinforce the findings seen in the literature that ANN ensembles can improve the performance of ANNs and in general Ensemble averaging is more effective than approximate the Bayesian model average via a BNN. Unfortunately, Table 3 seems to provide contradictory evidence to the hypothesis that BNNs would outperform comparable CNNs. This can be quantified using a one-sided paired t-test between the fold wise validation accuracies. The t-test is paired since each fold-wise validation estimate is trained on the same training fold and tested on the same fold. The central limit theorem provides justification as to why the fold-wise validation accuracies would be normally distributed. However, a small caveat of using a t-test here is that the fold wise validation estimates are correlated, this was previously touched upon in the Literature Review, and thus violate the independence assumption of the t-test. However, in this case this leads to the variance being underestimated, hence the t statistic will have a propensity to appear more extreme than it actually is. This results is a decrease is statistical size and an accompanied increase in statistical power. Thus, scepticism over the validity of this test is warranted in the situation that the the null is rejected, since failing to reject the null is performed more conservatively under these violated assumptions [43]. The null and alternative hypotheses, where ACC denotes accuracy, are summarised below:

$$H_0: \quad \text{ACC(CNN)} = \mathbb{E}(\text{ACC}(\text{CNN}, \mathcal{D}^{(i)})) \geq \mathbb{E}(\text{ACC}(\text{BNN}, \mathcal{D}^{(i)})) = \text{ACC(BNN)}$$
$$H_1: \quad \text{ACC(CNN)} = \mathbb{E}(\text{ACC}(\text{CNN}, \mathcal{D}^{(i)})) < \mathbb{E}(\text{ACC}(\text{BNN}, \mathcal{D}^{(i)})) = \text{ACC(BNN)}$$

The results of this paired one-sided t-test are shown in Table 4.

Model Pairing	t	p
BNN/CNN	-1.4222	0.886
BNN/CNN Ensemble	-2.5422	0.9681

Table 4: t-test results

The results from the t-test shown in Table 4 suggest that in both scenarios, wether comparing the single ANNs or ensemble ANNs, one fails to reject the null hypothesis that the BNN preforms equally if not worse than the CNN. This may suggest that audio classification problems conform to the manifold hypothesis more than was originally anticipated. It could also signify that the loss function associated with these types of problems do not have flat local minima, the local minima may instead resemble sharp peaks. As a result the VPD will aggregate many sub optimal hypothesis in its calculation.

6 Conclusion

The evidence collected throughout the duration of this investigation has been insufficient to reject the null hypothesis that a BNN will outperform a comparable CNN when applied to the task of audio classification. This investigation did however reinforce the current literature's findings that aggregation via deep ensembles outperforms variational approximations of the Bayesian model average generated via BNNs. The ineffectiveness of BNNs throughout this investigation may suggests that typical audio datasets conform to the manifold hypothesis more so than was previously understood. Additionally, the findings of this work may indicate properties of the loss functions associated with such problems. Further work should explore these ideas and compare a more diverse set of Bayesian architectures, including but not limited to a full BNN.

7 Bibliography

[1] Geoffrey Grimmett and David Stirzaker. **Probability and Random Processes**. Oxford university press, 2001. pages 10

[2] Jeffrey S Rosenthal. **First Look At Rigorous Probability Theory, A**. World Scientific Publishing Company, 2006. pages 10, 17

[3] Larry Wasserman. **All of Statistics: A Concise Course in Statistical inference**, volume 26. Springer, 2004. pages 10, 12

[4] G Alastair Young, Thomas A Severini, George Albert Young, RL Smith, Robert Leslie Smith, et al. **Essentials of Statistical Inference**, volume 16. Cambridge University Press, 2005. pages 10

[5] Richard T Cox. Probability, frequency and reasonable expectation. **American journal of physics**, 14(1):1–13, 1946. pages 10

[6] Christopher M Bishop. Pattern recognition and machine learning. **Machine learning**, 128(9), 2006. pages 10, 22

[7] Edwin T Jaynes. **Probability Theory: The Logic of Science**. Cambridge university press, 2003. pages 10, 11

[8] Devinderjit Sivia and John Skilling. **Data Analysis: A Bayesian Tutorial**. OUP Oxford, 2006. pages 11

[9] David JC MacKay and David JC Mac Kay. **Information Theory, Inference and Learning algorithms**. Cambridge university press, 2003. pages 11

[10] Kenneth Lange. **Numerical Analysis for Statisticians**. Springer Science & Business Media, 2010. pages 12, 17

[11] 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, 2017. pages 12, 13

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. **Deep Learning**. MIT Press, 2016. `http://www.deeplearningbook.org`. pages 12, 13, 19, 23, 24, 25, 30, 31, 34, 35, 43

[13] Walter Rudin et al. **Principles of mathematical analysis**, volume 3. McGraw-hill New York, 1964. pages 13

[14] Kevin P Murphy. **Machine Learning: A Probabilistic Perspective**. MIT press, 2012. pages 13, 15

[15] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In **International Conference on Machine Learning**, pages 1613–1622. PMLR, 2015. pages 14, 37, 38, 39, 40, 41, 42, 45, 64

[16] Yaser S Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. **Learning from data**, volume 4. AMLBook New York, NY, USA:, 2012. pages 15, 16, 17, 18, 19, 21, 28, 29, 42

[17] Richard S Sutton and Andrew G Barto. **Reinforcement learning: An introduction**. MIT press, 2018. pages 15

[18] Pedro Domingos. A few useful things to know about machine learning. **Communications of the ACM**, 55(10):78–87, 2012. pages 16

[19] Vladimir Vapnik. **The nature of statistical learning theory**. Springer science & business media, 2013. pages 17

[20] Sheldon Jay Axler. **Linear algebra done right**, volume 2. Springer, 1997. pages 18

[21] Aurélien Géron. **Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems**. O'Reilly Media, 2019. pages 20, 24, 25, 32, 33, 34, 35, 36

[22] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. **The elements of statistical learning**, volume 1. Springer series in statistics New York, 2001. pages 22

[23] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives, 2014. pages 23

[24] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In **International conference on machine learning**, pages 448–456. PMLR, 2015. pages 35, 36

[25] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. **Journal of Machine Learning Research**, 15(56):1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`. pages 36

[26] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. pages 42

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. **Advances in neural information processing systems**, 25:1097–1105, 2012. pages 43

[28] Andrew Gordon Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. **arXiv preprint arXiv:2002.08791**, 2020. pages 43, 44, 45

[29] Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep ensembles: A loss landscape perspective. **arXiv preprint arXiv:1912.02757**, 2019. pages 44, 45

[30] Karol J Piczak. Esc: Dataset for environmental sound classification. In **Proceedings of the 23rd ACM international conference on Multimedia**, pages 1015–1018, 2015. pages 46

[31] John C Steinberg. Positions of stimulation in the cochlea by pure tones. **The Journal of the Acoustical Society of America**, 8(3):176–180, 1937. pages 48

[32] Meinard Müller. **Fundamentals of music processing: Audio, analysis, algorithms, applications**. Springer, 2015. pages 48, 49, 50, 51

[33] Fredric J Harris. On the use of windows for harmonic analysis with the discrete fourier transform. **Proceedings of the IEEE**, 66(1):51–83, 1978. pages 50

[34] Brian McFee, Alexandros Metsai, Matt McVicar, Stefan Balke, Carl Thomé, Colin Raffel, Frank Zalkow, Ayoub Malek, Dana, Kyungyun Lee, Oriol Nieto, Dan Ellis, Jack Mason, Eric Battenberg, Scott Seyfarth, Ryuichi Yamamoto, viktorandreevichmorozov, Keunwoo Choi, Josh Moore, Rachel Bittner, Shunsuke Hidaka, Ziyao Wei, nullmightybofo, Darío Hereñú, Fabian-Robert Stöter, Pius Friesch, Adam Weiss, Matt Vollrath, Taewoon Kim, and Thassilo. librosa/librosa: 0.8.1rc2, May 2021. URL `https://doi.org/10.5281/zenodo.4792298`. pages 51, 54, 57

[35] Hohyub Jeon, Yongchul Jung, Seongjoo Lee, and Yunho Jung. Area-efficient short-time fourier transform processor for time–frequency analysis of non-stationary signals. **Applied Sciences**, 10(20):7208, 2020. pages 52

[36] Stanley Smith Stevens, John Volkmann, and Edwin Broomell Newman. A scale for the measurement of the psychological magnitude pitch. **The journal of the acoustical society of america**, 8(3):185–190, 1937. pages 54

[37] Douglas O'shaughnessy. **Speech Communications: Human And Machine (ieee)**. Universities press, 1987. pages 54

[38] Jaehun Kim. Urban sound tagging using multi-channel audio feature with convolutional neural networks. **Proceedings of the Detection and Classification of Acoustic Scenes and Events**, 2020. pages 56, 57

[39] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, **Proceedings of the 36th International Conference on Machine Learning**, volume 97 of **Proceedings of Machine Learning Research**, pages 6105–6114. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/tan19a.html`. pages 57

[40] Loris Nanni, Gianluca Maguolo, Sheryl Brahnam, and Michelangelo Paci. An ensemble of convolutional neural networks for audio classification. **Applied Sciences**, 11(13):5796, 2021. pages 58, 60

[41] Md Mohaimenuzzaman, Christoph Bergmeir, Ian Thomas West, and Bernd Meyer. Environmental sound classification on the edge: A pipeline for deep acoustic networks on extremely resource-constrained devices. **arXiv preprint arXiv:2103.03483**, 2021. pages 59, 60, 63

[42] Karol J Piczak. Environmental sound classification with convolutional neural networks. In **2015 IEEE 25th international workshop on machine learning for signal processing (MLSP)**, pages 1–6. IEEE, 2015. pages 62

[43] Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. **Neural Computation**, 10(7):1895–1923, 1998. doi: 10.1162/089976698300017197. pages 71