UNIVERSITY OF OULU

FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Saku Antikainen
Lassi Tölli
Olli Paananen

FREE ENERGY PRINCIPLE INSPIRED IMAGE RECOGNITION WITH CONVOLUTIONAL NEURAL NETWORKS

Bachelor's Thesis
Degree Programme in Computer Science and Engineering
June 2022

Antikainen S., Tölli L., Paananen O. (2022) Free Energy Principle Inspired Image Recognition with Convolutional Neural Networks. University of Oulu, Degree Programme in Computer Science and Engineering, 46 p.

ABSTRACT

Since its inception, the field of neuroscience has studied the way the human brain perceives and learns about its environment. Several theories have been created in an effort to understand these phenomena and few have garnered as much interest as Karl Friston's free energy principle (FEP) [1] . The free energy principle states that any self-organizing system that is at equilibrium with its environment must minimize its free energy. The principle is essentially a mathematical formulation of how adaptive systems (that is, biological agents, like animals or brains) resist a natural tendency to disorder. Friston's principle provides a framework that explains not only how the brain functions, but how any stable system organizes itself. Unsurprisingly a theory of this magnitude has created a lot of debate and received fair share of criticism. Whether or not Friston's principle is correct or not, it has been proven to be a functional framework in the context of machine learning.

The goal of this thesis is to provide an example of a practical implementation of the FEP in the form of a Bayesian Neural Network (BNN), execute image classification on it, and compare its performance to another neural network, Convolutional Neural Network (CNN). Both of these networks are trained with a few different datasets and we are comparing the accuracies and training times of these networks.

We begin with an introduction to the key concepts that will help the reader to better understand the topic of this paper. We then present some related work on this topic and continue to introduce the architecture of the CNN and BNN models. In the subsequent section we showcase the results of the training for both the BNN and the CNN. We also provide an analysis of the results of the thesis and discuss possible future work. The final conclusions section contains a summary of the project. The results from the experiments suggest that a CNN is overall more accurate in classifying images. This does not mean that a BNN is useless since the BNN can express uncertainty which is something a CNN is incapable of.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS AND SYMBOLS

CNN	convolutional neural network
BNN	Bayesian neural network
FEP	free energy principle
p	Probability
D_{kl}	Kullback–Leibler divergence
θ	Parameter vector
σ^2	Variance
μ	Mean
$\mathbb{E}$	Expectation

1. INTRODUCTION

This chapter is divided into three sections. Section 1.1 will explain Friston's free energy principle and how minimizing free energy relates to machine learning in general. Section 1.2 introduces predictive coding and explains how it relates to our thesis through variational inference. Section 1.3 will give a brief introduction to how neural networks function and briefly compare the differences between the convolutional neural networks (CNN) and the Bayesian neural networks (BNN).

1.1. The Free Energy Principle

The free energy principle (FEP) is a theory in cognitive science that was introduced by the neuroscientist Karl Friston. The FEP states that any self-organizing system (i.e. any dynamical system, and therefore any living or cognitive entity) equipped with a Markov blanket – a statistical separation between internal and external states – can be interpreted as performing Bayesian inference upon the surrounding environment, such that its internal states come to encode probabilistic beliefs about the external environment [2]. The FEP covers both living and non-living systems but our thesis will explain the FEP from the perspective of a living system i.e the human brain. To understand the concept of free energy better, we can divide reality into two different states: the internal state and the external state. The internal state depicts all the things happening within our brain and the external state depicts everything happening outside of our brain (see Figure 1). There are two ways in which our brain and the external world can communicate with each other. First one is through our senses. Our brain receives information from world by observing it using our senses, for example by seeing or hearing. The second one is by actions, for example moving our hand. The idea is that our brain tries to model the world as accurately as possible from the observation data it gets.

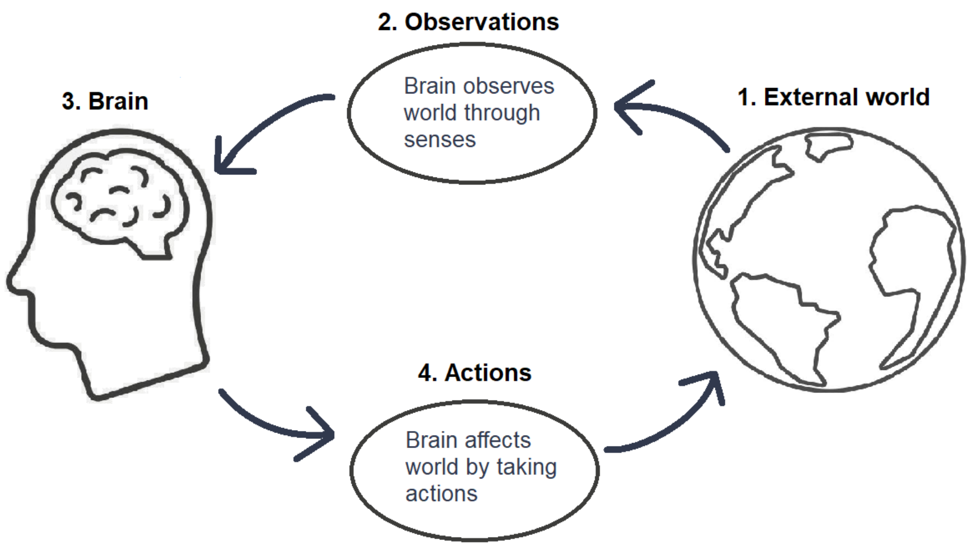
2. Observations
Brain observes world through senses
3. Brain
1. External world
4. Actions
Brain affects world by taking actions

Figure 1. Free energy principle states

According to the FEP, the brain models the world by minimizing free energy. The brain predicts the next states of the external world using all the input data it has gotten through observations. Mathematically speaking, minimizing free energy means minimizing the prediction error between these predicted states and the actual states of external world [3]. We can think of this error as "surprise"that the brain minimizes by correcting its model of the world.

Accuracy is the expectation of log-likelihood with respect to the approximate posterior, which represents reconstruction of the observation with the approximate posterior. Complexity is the Kullback-Leibler divergence between the approximate posterior and the prior, which serves to regularize the model. Importantly, in maximizing the lower bound, the interplay between these two terms characterizes how the model behaves in learning and prediction [4].

1.2. Predictive Coding

Predictive coding is a theory in computational neuroscience which proposes that the core function of the brain is to minimize prediction error between predicted inputs and actual inputs that the brain received [5]. The theory also states that the brain is constantly generating and updating a model of its environment using the comparison results in the prediction error. The theory of predictive coding has its roots in Hermann von Helmholtz's theory of unconscious inference which states that "perception is indirectly influenced by inferences about current sensory input that make use of the perceiver's knowledge of the world and prior experience with similar"[6]. While Predictive coding as a neuroscientific theory originates in the 1980s and 1990s from the work of people such as Rao & Ballard, it was first developed into its modern mathematical form of a comprehensive theory of cortical responses in the mid 2000s by Karl Friston. [5]. The initial works of Rao & Ballard and Friston have been significantly improved on with newer extended theoretical and mathematical models of predictive coding and empirical testing [5]. However, the predictive coding theory still has many unsolved problems that don't seem to fit with the current framework [5]. Research on predictive coding has not yet provided us with a unifying theory for cortical function but what it has produced is a mathematical framework in the form of the variational inference algorithm.

Variational inference itself is a term used to describe a family of methods which has been developed in both machine learning and statistics. K. Friston [7] recognized that the predictive coding algorithm could be cast as an approximate Bayesian inference processed based upon Gaussian generative models. In general, variational inference approximates an intractable inference problem with a tractable optimization problem. We can better demonstrate variational inference through an example.

Let us assume that we have weights w and data $\mathcal{D}$. We wish to find out the value for our weights for the current data. We can get this conditional probability using the Bayes' theorem $p(w|\mathcal{D}) = \frac{p(\mathcal{D}|w)p(w)}{p(\mathcal{D})}$ The prior in the denominator is also known as the normalizing factor and it can be written as p($\mathcal{D}$) = $\int p(\mathcal{D}, w)\, dw$. In general, there is no efficient algorithm for calculating this integral meaning it is intractable. The intractable denominator makes the calculation of the conditional probability into an intractable problem. This is where variational inference steps in. Instead of trying to solve the conditional probability we can approximate it with another posterior $q_\theta(w|\theta)$. q is a variational distribution than can be chosen by the person making the approximation. For this example we will assume $q_\theta(w|\theta)$ has a Gaussian distribution. The parameters $\theta = \{\mu, \Sigma\}$ are the mean μ and variance Σ of the Gaussian distribution. Now our task is to minimize the divergence between the true and approximate posteriors with respect to the Gaussian distributions mean and variance. This problem can be written as

$$\theta^{opt} = \underset{\theta}{\operatorname{argmin}}\ \mathbb{D}_{kl}[q_\theta(w|\theta)||p(w|\mathcal{D})]$$

where $\mathbb{D}_{kl}[q|p]$ is the Kullback–Leibler divergence between the true and approximate distributions. By applying Bayesian rules to the true posterior of the problem and using some properties of logarithms one can rewrite the equation to the form of free energy

$$\mathcal{F}(\mathcal{D}, \theta) = D_{kl}[q(w|\theta)||p(w)] - \mathbb{E}_{q(w|\theta)}[\log p(\mathcal{D}|w)] \tag{1}$$

$\mathcal{F}(\mathcal{D}, \theta)$ = *Free energy*
$D_{kl}[q(w|\theta)||p(w)]$ = *The complexity term*
$\mathbb{E}_{q(w|\theta)}[\log p(\mathcal{D}|w)]$ = *The Accuracy term*
$q(w|\theta)$ = *An approximation of the true distribution*
$p(w)$ = *The prior probability*
$(\mathcal{D}|w)$ = *The data dependent conditional probability*

Accuracy is the expectation of log-likelihood with respect to the approximate posterior, which represents reconstruction of the observation with the approximate posterior. Complexity is the Kullback-Leibler divergence between the approximate posterior and the prior, which serves to regularize the model. Importantly, in maximizing the lower bound, the interplay between these two terms characterizes how the model behaves in learning and prediction [4]. From this point forward one could start optimizing the equation and there are many ways this can be done [8].

Even though this form of the equation can already be optimized, one can further simplify the equation by making two additional assumptions [5]. Firstly we need to assume that the generative model takes a Gaussian form meaning

$$p(\mathcal{D}, w) = p(\mathcal{D}|w)p(w) = \mathcal{N}(\mathcal{D}; f(\theta_1 w), \Sigma_1)\mathcal{N}(w; g(\theta_2\overline{\mu}), \Sigma_2)$$

The mean of the likelihood Gaussian distribution is assumed to be some function f of the weights w, which can be parameterized with parameters θ, while the mean of the prior Gaussian distribution is set to some function g of the prior mean $\overline{\mu}$. The variances of the two gaussian distributions of the generative model are denoted Σ_1 and Σ_2. Secondly, we assume that the variational posterior is a dirac-delta distribution $q(w|\theta) = \delta(w - \mu)$. Now that we have made these assumptions, we can decompose the free energy equation to 'Entropy' and 'Energy' terms [3]

$$F(\mathcal{D}, \theta) = \mathbb{E}_{q(w|\theta)}[ln(q(w|\theta))] + \mathbb{E}_{q(w|\theta)}[ln(p(\theta, w))]$$

where the first term is 'Entropy' and the second term is 'Energy'. Because of the second assumption we made, we can ignore the 'Entropy' term and since the entropy of a dirac-delta distribution is zero. Now we can write 'Energy' in the following way.

$$\begin{aligned}
\mathbb{E}_{q(w|\theta)}[\ln(p(\theta, w))] &= \mathbb{E}_{\delta(\theta-\mu)}[\ln(\mathcal{N}(\mathcal{D}; f(\theta_1 w), \Sigma_1)\mathcal{N}(w; g(\theta_2\overline{\mu}), \Sigma_2))] \\
&= \ln(\mathcal{N}(\mathcal{D}; f(\theta_1 w), \Sigma_1) + \ln(\mathcal{N}(w; g(\theta_2\overline{\mu}), \Sigma_2)) \\
&= -\frac{(w - f(\mu, \theta_1))^2}{2\Sigma_1} - \ln 2\pi\Sigma_2 - \frac{(\mu - g(\overline{\mu}, \theta_2))^2}{2\Sigma_2} - \ln 2\pi\Sigma_1 \\
&= -\frac{1}{2}[\Sigma_1^{-1}\epsilon_\theta^2 + \Sigma_2^{-1}\epsilon_w^2 + \ln 2\pi\Sigma_1 + \ln 2\pi\Sigma_2]
\end{aligned} \tag{2}$$

In the second to last line of the Equation (2) we have two similar terms so we note these terms with new variables $\epsilon_w = \mu - g(\overline{\mu}, \theta_2)$ and $\epsilon_\theta = \theta - f(\mu, \theta_1)$. These new variables are the prediction errors of the equation. From this point forward we can begin the optimization of the variational free energy by using gradient descent. Using gradient descent we can update the variables $(\mu, \theta_1, \theta_2)$. Gradient descent is explained in Section 1.3.

1.3. Neural Networks

A neural network is a machine learning model and a key concept in deep learning. A neural network is formed from a given amount of input nodes, hidden layers with a given number of nodes within them, and an output layer with a given amount of output nodes that represent the number of possible outcomes the network is expected to produce (see Figure 2). A node is essentially a mathematical function that gives a specific output, or 'activation' with the given inputs, which are all affected by a factor, also known as weight of the connection. All the nodes from a layer are connected to the nodes of the previous layer, with all connections having individual weights assigned to them. These weights are what make certain nodes activate when presented with data they can recognize, eventually giving a true output at the output nodes.

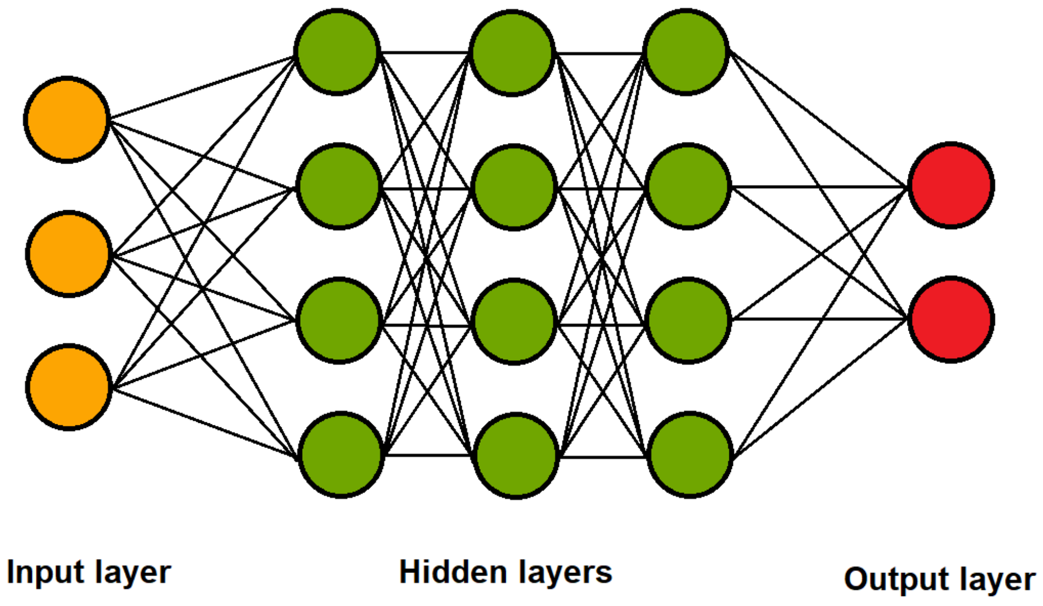

Figure 2. Schematic layer structure in neural networks

A neural network can learn the correct values for the weights in the process of training. In training, the neural network is given a sample from a set of data that it will use to adjust the weights. These sets of data including the corresponding expected outputs are called datasets. Datasets are separated into smaller subsets. Each of these subsets are used only for a single task for example training data is used only during the training process.

For instance, imagine a CNN that is trying to recognize handwritten numbers from images. At the time of the training the network, it is shown images of handwritten numbers (training data), and it starts guessing which numbers they are. After each image, the network generates different levels of activation for each node, these are then adjusted using the weights of the connections, in a way that all the wrong outputs are adjusted towards a smaller activation, while the right output is adjusted towards a larger activation. This is done through a process called backpropagation. Backpropagation is an algorithm which goes through the neural network starting from the output nodes and adjusting the weights of the connections by computing the gradient of a loss function with respect to the weights. The loss of a neural network is a metric which we can use to evaluate the performance of the neural network during training. A loss function is method that is used to quantify the performance of the neural network by summing up the overall error between the observed and predicted values. Loss functions can roughly be divided into two categories, regression loss and classification loss. In classification our goal is to predict a discrete class output whereas in regression the output is continuous in nature. The training process can be considered complete when additional observations of data does not significantly reduce the amount of error. After the training, the performance of the network can be evaluated using testing data which is one of the subsets from the larger dataset. If the training has been successful, the neural network should be able accurately categorize handwritten numbers from the testing data. The accuracy of the networks predictions is a metric that can be used to measure the networks performance after training.

Inspired by predictive coding, we decided to research the inner workings of Bayesian neural networks and compare their performance to the performance of more traditional neural networks. Therefore, our thesis will mainly focus on two different types of neural networks. The chosen neural networks are a Convolutional neural network (CNN) and a Bayesian Convolutional neural network (BNN). The most notable difference between these two neural networks is that a CNN has single floating-point values as weights, whereas in a BNN, these values are replaced with probability distributions. A more detailed description of both networks can be found in Chapter 3. For this study, we chose to compare the performance of a CNN and a BNN on specific learning tasks. This choice was motivated by the fact, that CNNs are arguably the most popular in tasks like image classification and have been proven to have great performance in them. [9]. A BNN was chosen for this study because of its connection to predictive coding.

1.4. The Relationship Between Predictive Coding and BNN

The connection between predictive coding and the BNN can be found in variational inference. BNNs can use variational inference to infer distributions for the model weights. A more in depth explanation of the BNN can be found in Section 3.2. It is possible to make the same assumptions that were made for Equation 2 so that the BNN functions in a way that resembles predictive coding [10][11]. However, Bayesian inference needs to be adjusted for deep learning. The main reason for this is that approximating the posterior in BNNs can be fairly computationally expensive. The use of Gaussian approximating distributions increases the number of model parameters considerably, without increasing model capacity by much. This increased amount in model parameters makes the Bayesian neural network scale poorly to large dataset and network sizes [11]. Several different approaches have been proposed to make Bayesian inference more suitable for deep learning purposes, such as probabilistic backpropagation [11] and Bayes by Backprop [12]. Our BNN utilizes Bayes by Backprop and an in depth explanation for it can be found in Section 3.2.1. Using the Bayes by Backprop algorithm has the added benefit that the algorithm is very similar to the classical training loop for point estimate deep learning, most techniques used for optimization in deep learning are straightforward to use for Bayes-by-Backprop [8].

2. RELATED WORK

In this section six papers relating to the subject matter of this thesis will be summarized to give readers a more complete picture of the current state of the field.

2.1. Applications of the Free Energy Principle to Machine Learning and Neuroscience

In his thesis, Millidge comprehensively explores the free energy principle and its applications in both machine learning and neuroscience [13]. The thesis is separated into three sections.

The first section contains chapters 2 and 3. In Chapter 2, Millidge gives a detailed overview of the free energy principle, starting from first principles, and including a discussion of the mathematical assumptions and he provides some of his opinions on the philosophical nature of the theory and its potential utility. He also gives a brief walk-through of discrete state space active inference [14] which is the focus of the 'scaling up' work in Chapter 4. In Chapter 3, Millidge deals principally with models of perception and predictive coding. He begins by giving a brief overview and mathematical walkthrough of predictive coding theory. He then covers in depth two contributions to the theory of predictive coding. First, he presents work where he scales up and empirically test the performance of large scale predictive coding networks on machine learning datasets. He also clarifies the relationship between predictive coding and other known algorithms such as Kalman filtering. Secondly, Millidge discusses relaxing various relatively un-biologically plausible aspects of the predictive coding equations, such as the need for symmetric forward and backwards weights, the necessity of using nonlinear derivatives in the update rules, and the one-to-one error to value neuron connectivity required by the standard algorithms. Then, in Chapters 4 and 5, Millidges presents his work on the applications of the free energy principle to questions of action selection and control. Chapter 4 focuses predominantly on scaling up active inference methods to achieve results comparable to those achieved in the deep reinforcement learning literature, while Chapter 5 takes a more abstract and mathematical approach and investigates in depths the mathematical origin of objective functionals which combine both exploitatory and exploratory behaviour – an approach which has the potential to finesse the exploration-exploitation dilemna [15].

Finally Millidge discusses the the most important and solvable questions within his field and he argues that his work has answered some of these questions. In Millidge's opinion the greatest question still remains which is how can credit assignment be implemented in the brain? And, specifically, whether and how the brain can implement the backpropagation of error algorithm.

2.2. Deep Predictive Coding Networks For Video Prediction And Unsupervised Learning

This study is about implementing a predictive neural network that tries predicts future frames in a video [16]. It is based on a previous work on the same topic [17], but this study adds a predictive coding aspect into it. In the second chapter the structure of the predictive neural network is introduced. Their model is called "PredNet"and the architecture of this model is based on the predictive coding model proposed by Rao and Ballard in 1999 [18]. The network is built by modules that are each divided into four layers. There is a representation layer $\mathbf{R_l}$, which utilises a recurrent convolutional network to generate a prediction of the next frame. The second layer is the prediction layer, and it is called $\mathbf{A_{l1}}$. The third layer $\mathbf{A_{l2}}$ is the layer for the actual frames. The final layer $\mathbf{E_l}$ outputs the difference between predicted frame and the actual frame. The difference is calculated using the layers $\mathbf{A_{l1}}$ and $\mathbf{A_{l2}}$. This error is then passed to a $\mathbf{A_{l2}}$ and incorporated into the next input $\mathbf{A_{l2+1}}$.

In the third chapter the authors tested this model by training it with a data consisting of synthetic images. The data contained sequences of ten frames of face rotating to a certain direction. The model attempted to predict the next frame in each of these sequences and it managed to do it with a decent accuracy. For comparison, this same learning task was carried out with a variant of the Ladder Network [19] and a standard autoencoder. When comparing the results, it turned out that this PredNet model was more accurate on predicting future frames than the standard autoencoder or the Ladder Network.

The next step was to test the model with real-world images. The training data consisted of raw videos recorded with a car-mounted camera. The data was further divided into a few different categories and sampled to sequences of 10 frames. The PredNet model predicted future frames fairly accurately. The same learning task was carried out also with CNN-LSTM Encoder-Decoder and with a very simple model that simply copies the last frame to be the next prediction. The CNN-LSTM (Long Short-Term Memory) Encoder-Decoder is a control model that shares the overall architecture and training scheme of the PredNet, but that sends forward the layer-wise activations $\mathbf{A_l}$ rather than the errors $\mathbf{E_l}$. Once again, the accuracy was best with PredNet model. In conclusion, this study demonstrates that predictive coding neural networks can be relatively effective in image classification-like scenarios.

2.3. Hands-On Bayesian Neural Networks -- a Tutorial for Deep Learning Users

The tutorial by Jospin et al. [20] gives an overview of Bayesian neural networks.

In the first three sections of the paper Jospin et al. give a clear and concise explanation of what exactly is a Bayesian neural network, why a Bayesian model is being researched and what are its benefits. Jospin et al. define the BNN as a stochastic artificial neural network trained using Bayesian inference. Stochasticity means that the network has either stochastic activation functions or stochastic weights. The Bayesian model is being researched because the modern deep learning methods function as black boxes and the uncertainty associated with their predictions is often challenging to quantify. One of the main benefits of the BNN is that it provides a natural approach to quantify uncertainty. The fourth and fifth section go more in detail about the design choices that need to be made when building a Bayesian neural network. The fourth section shows how to design the stochastic model for the network and how certain choices affect degree of supervision of the model. The fifth section gives five algorithms that can be used to calculate an approximation for the solution of the Bayesian theorem for our data. The final section gives ways to evaluate the performance of the Bayesian neural network. This mainly happens through the assessment of the calibration curve with various methods. In addition to the tutorial Jospin et al. provide supplementary material where three problems are solved using different variants of the Bayesian neural network architecture. The source codes for these solutions are also provided through GitHub.

2.4. A Comprehensive Guide to Bayesian Convolutional Neural Network with Variational Inference

Convolutional neural networks are widely used in image classification, and they are generally presumed to perform well in recognizing patterns in images. In fact, best CNNs can already outperform humans in this area [21]. Even though CNNs are proven to be good at image classification tasks, they have a number have couple of major downsides. The first downside is that training CNN with a small dataset tend to cause overfitting. This means that the model is too detailed to this specific data and thus might not work that well with more general data. This causes the network to make over-confident decisions. The second problem CNNs have is the lack of an uncertainty measure in its predictions. This study tries to overcome these problems by introducing Bayesian learning and the usage of variational inference in convolutional neural networks.

The basic idea of this approach is to replace the single-point estimated weights in a CNN with probability distributions. The Bayesian approach of using distributions allows the model to express uncertainty and averages out the parameters, which helps preventing overfitting.

The architecture for the Bayesian CNN used in this study is based on an algorithm called Bayes by Backprop [22] [12], (see Section 3.2.1). Since the true probability distributions $p(w|D)$ in Bayesian inference are intractable this study approximates them with variational probability distributions $q_\theta(w|D)$. These approximated distributions contain the same parameters as a Gaussian distribution, mean μ and variance σ^2. The second chapter focuses on key concepts. Most of the key concepts are already covered in our thesis and we skip the details for now. See Chapter 3 for more detailed explanations of the models.

How does one replace the single-point estimates with probability distributions? A regular CNN calculates the values for the weights with a single convolutional operation but in order to construct variational posterior probability, we need two separate values, mean and variance. With the help of local reparametrization trick (see Section 3.2.2) the network can calculate values for mean and variance form the outputs of our two convolutional operations. The output of the first convolutional operation is treated as in a regular CNN to provide a value for the mean μ. In the second convolutional operation we will learn the value for α which can be used with the mean to calculate the value for variance via the formula: $\sigma^2 = \alpha\mu^2$. These parameters define the variational distribution $q_\theta(w|D)$.

$$q_\theta(w|D) = N(\mu, \sigma^2)$$

The last few chapters of this study document the tests the author carried out with their Bayesian and Non-Bayesian models. They applied these models to two architectures (AlexNet and LeNet-5) and compared their performances in different scenarios. They tested these networks with learning tasks related to image recognition, image super-resolution and Generative Adversarial Networks [23]. Image super resolution is the task of recovering a high-resolution image from a given low-resolution image. The results show that Bayesian CNN is comparable with a regular CNN. The BNN achieved a comparable validation accuracy to the CNN in image classification. The BNN was also able to achieve results that are comparable in terms of the number and the quality of the image generated in the image super-resolution task. In the final experiment the authors created a Bayesian DCGAN [24] that they then compared to the original DCGAN. The Bayesian had a significantly higher loss than the authors anticipated but according to the authors, there is no comparison that can be drawn from the results of the two networks. Since GANs are difficult to anticipate just by the loss number, a comparison cannot be made.

2.5. Practical Deep Learning with Bayesian Principles

One of the greater issues that the Bayesian neural networks face is the lack scalability. The task of accurately approximating a posterior distribution is very difficult which is the reason why Bayesian neural networks scale poorly to large datasets such as ImageNet. In this study Osawa et al. demonstrate practical training of deep networks with natural-gradient variational inference. They do this by implementing a new optimizer called the Variational Online Gauss-Newton (VOGN) method which was developed by Khan et al. [25]. The VOGN requires lower memory, computation, and implementation effort than existing VI methods and the VOGN takes a form similar to the more commonly known ADAM optimizer which allows Osawa et al. to borrow existing deep learning techniques to further improve the performance of their training. The first method they use is batch normalization in which BatchNorm layers are inserted between neural network layers. These layers help stabilize each layer's input distribution by normalizing the running average of the inputs' mean and variance. The second method they used was data augmentation and more specifically random cropping and horizontal flipping. Unlike the batch normalization, data augmentation could not be implemented without making any changes. Using data augmentation resulted in slightly worse performance than expected, and it also affected the calibration of the resulting uncertainty.

In the experiment portion of the paper, they compare the performance of three separate CNN architectures, these architectures being the AlexNet, LeNet-5, and ResNet-18. They found that the networks that use VOGN were the best or tied on best on ten of the fifteen metrics that they used for performance evaluation. In conclusion, they found that the networks that used VOGN had accuracies and convergence rates comparable to the networks that use SGD and Adam. In addition to this, the VOGN retains a well calibrated uncertainty and good performance on out-of-distribution data.

2.6. Bayesian Convolutional Neural Networks with Variational Inference

In his report Yi-Pei Chan reproduced the Bayesian LeNet model from [21]. He tested his network on image classification task with the MNIST, CIFAR-10, and CIFAR-100 datasets. Chan then compared the performance of his Bayesian LeNet and the original LeNet networks to the to the results found in [21]. Chan's models performed worse than the models in the original study[21] and Chan speculates that this might be because he needs to make initialization for the mean and variance parameters in the variational posterior probability distribution, and bad initialization values may result in longer convergence time to the optimum solution.

3. NEURAL NETWORKS

In this section we will briefly explain how a convolutional neural network functions and introduce the CNN model we are using in this thesis. We will also explain how the Bayesian neural network functions, introduce its architecture, and discuss the choices that were made when implementing the network.

3.1. Convolutional Neural Network

In Chapter 1, we gave a quick rundown on how neural networks function. The type of the described neural network is called a feedforward neural network. The two features to remember from this network are that data only travels to one direction and that all the nodes from a layer are connected to the nodes of the previous layer. This is a good general-purpose architecture that has its strengths and weaknesses. A major weakness of a feedforward neural network is the lack of scalability when classifying images.

For example, assume we have a 100x100 pixel black and white image. In a feedforward network, the grid of pixels in the image would be converted into a 10 000-input node column and these nodes would be connected to each of the nodes in the first hidden layer. Each of these connections would have a weight that needs to be estimated. If the first hidden layer has ten nodes, this means that we need to estimate a total of 10 000 weights * 10 nodes = 100 000 weights with each backpropagation step. This is a significant computational load even though a 100x100 image is very small when compared to images encountered in real world applications, and a hidden layer with 10 nodes is relatively small.

The first layer after the input layer in a CNN is called the convolutional layer and it begins processing the image by applying a filter to the grid of pixels. As an example, we can determine a filter as a 3x3 array with random pixel values that are later updated using backpropagation. The filter is convolved with the input grid of pixels and the dot product of the convolution is inserted into a feature map. This gets repeated until the filter has convolved with every pixel in the image. A visual representation of this can be seen in Figure 3. After the convolutional layer comes a pooling layer where a new filter is applied to the feature map. This filter takes the maximum value out of the area where it is currently being applied and adds it to the input grid of values. After every pixel has been processed, the input grid values are converted into a column which is fed into the fully connected layer that has an activation function. The column that is fed to the third layer is significantly smaller than the column that we had in the example above, and that is why a CNN scales up much better than a feedforward network. From the fully connected layer we will get the networks' prediction for the class of the image. Just like with the feedforward network, we can now backpropagate in the CNN to update all of the weights after which the entire training process is repeated.

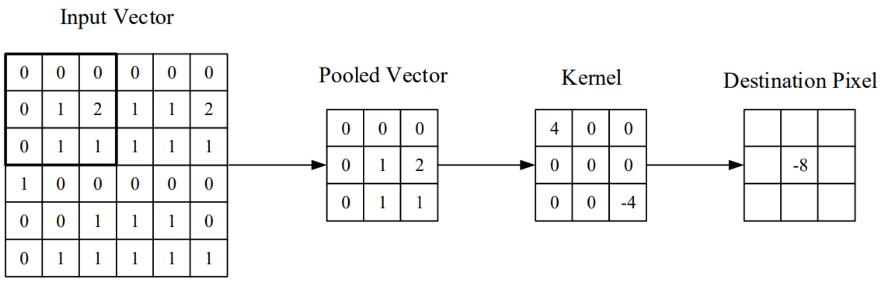

Figure 3. Convolutional layer filter, source [26]

This is about the simplest architecture a CNN can have, but it demonstrates well how even the larger networks' function. In addition to having good scalability, a CNN has other benefits. Firstly, it can utilize the correlation that happens in more complex images and secondly its performance is not severely affected by pixel shifts in the input images. The input images can also be subsampled or scaled without it affecting the end result. This means that we can further reduce the number of parameters that the network needs to evaluate. Overall, a network as simple as this can perform well and it can also be improved quite easily. One of the most common ways to improve a basic CNN is adding more depth to the network, which means adding additional convolutional and pooling layers before the fully connected layers. Networks that have a large number of layers are often referred to as deep convolutional neural networks and their architectures have been very successful in image recognition [27].

The CNN that we use in our thesis is called AlexNet [28]. AlexNet won the ImageNet Large Scale Visual Recognition Challenge in 2012 and it is considered to be a breakthrough network that brought significant amount of new interest in CNNs. We chose this network for two reasons. Firstly, the network is already implemented in the source code that we acquired from [21]. Secondly, it represents a well-known and performing network that has a very simple design when compared to the networks that currently perform the best. The Bayesian neural network in our thesis will also use the architecture of AlexNet.

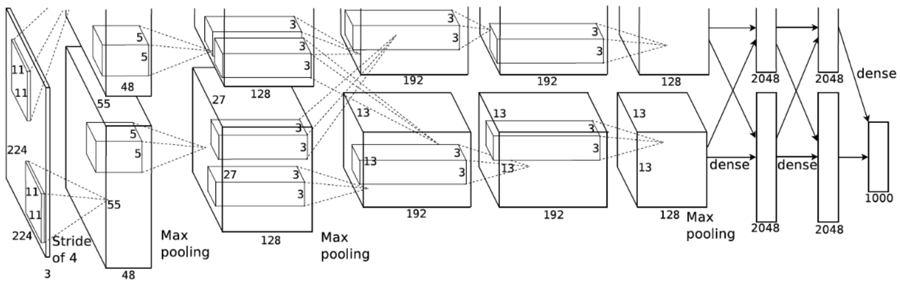

Figure 4. AlexNet architecture

3.2. Bayesian Neural Networks

How does the Bayesian Neural Network (BNN) model differ from the CNN model? In a BNN, the main difference can be found in the way the weights of the connections work. In a CNN the weights are calculated by computing the gradient of the loss function, and are given a floating-point value, whereas in a BNN, the weights are given a probability distribution, with the help of Bayes' theorem, (see Figure 5). Also all the values in the filters are replaced with a distribution. All of these distributions are Gaussian because it can be easily represented with just a two values, mean and variance. In practise this means that every time a CNN learns or updates a value in the weights, a BNN has to learn or update two values. This can be done by applying two consecutive convolutional operations. The output of the first convolutional operation is treated as in a regular CNN to provide a value for the mean μ. In the second convolutional operation we will learn the value for α. After that the value for variance can be calculated via the formula: $\sigma^2 = \alpha\mu^2$.

As stated in Section 1.2, calculating the probability with Bayes' theorem gives us an integral that is intractable, which is why a BNN uses variational inference to approximate the posterior with another distribution, trying to minimize the distance between them. Unlike a CNN, a BNN is also able to express uncertainty, because of the probability distributions. This means that in a way, a BNN has a way to say, "I don't know", when presented with data that is outside the scope of its training, e.g. an image of white noise instead of an object which the network has had training on.

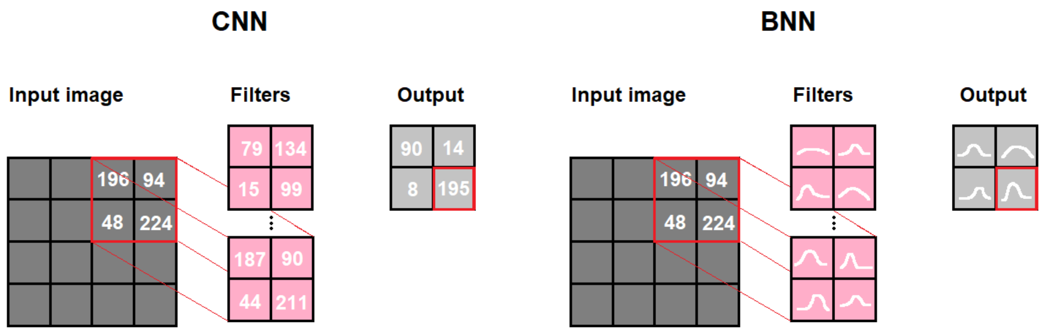

Figure 5. Demonstration of how CNN and BNN differ from each other with exemplary filter and input pixel values.

In this thesis we are going to utilize the neural network models that were implemented by Shridhad et al. in their paper [21]. The authors implemented three different CNN's that they then converted into bayesian neural networks. In this thesis we use the AlexNet versions of these CNN and BNN models and we focus on comparing the results from these two networks. Next we will explain three key concepts that are specific to the BNN. The first is *Bayes by Backprop* which is the way our BNN utilizes Variational inference. The second is a *local reparametrization trick* which significantly improves the performance of the BNN model. The last one, *weights pruning*, is a deep learning method that is used to reduce the amount of valued parameters in a model.

3.2.1. Bayes by Backprop

The main idea of the Bayes by Backprop method is to solve a fundamental problem associated with Baysian CNNs. In order to update the weights during the backpropagation process, one needs to calculate the gradients of the loss fucntion. The loss function our Bayesian model uses, also known as variational free energy, is intractable because the KL-divergence term can not be computed. The formula for our loss function is as follows

$$F(D,\theta) = D_{kl}(q_\theta(w|D)||p(w)) - \mathbb{E}_{q_\theta(w|D)}[\log(p(D|w))],$$

where D $=$ data, θ $=$ parameters mean and variance, $q_\theta(w|D)$ $=$ approximated probability distribution for weight w given data D, $p(w)$ $=$ the real probability distribution for weight w and $\mathbb{E}_{q_\theta(w|D)}[log(p(D|w))]$ is the accuracy term. By further expanding the KL-divergence term

$$D_{kl}(q_\theta(w|D)||p(w)) = \int q_\theta(w|D)\log\Big(\frac{q_\theta(w|D)}{p(w|D)}\Big)dw,$$

we can see that this term is calculated using an integral, which makes it intractable.

To solve this problem we have to use Bayes by Backprop, a method introduced by Blundell (2015) [12]. The main point of this method is to estimate the loss function with a tractable equation which makes the backpropagation process possible. Let us start by changing the form of our KL-divergence term. In general, expected value for a continuous random variable X is calculated in a following way

$$\mathbb{E}[X] = \int xf(x)dx$$

With this formula we change the form of our KL-divergence term.

$$\int q_\theta(w|D)\log\Big(\frac{q_\theta(w|D)}{p(w)}\Big)dw = \mathbb{E}_{q_\theta(w|D)}\Big[\log\Big(\frac{q_\theta(w|D)}{p(w)}\Big)\Big]$$

After that we can use the rule of logarithm to further modify this equation. The rule states that

equ

$$\log\Big(\frac{a}{b}\Big) = \log(a) - \log(b)$$

$$\Rightarrow \mathbb{E}_{q_\theta(w|D)}\Big[\log\Big(\frac{q_\theta(w|D)}{p(w)}\Big)\Big] = \mathbb{E}_{q_\theta(w|D)}[\log(q_\theta(w|D)) - \log(p(w))]$$

Now we can insert these experssions into the original loss function.

$$F(D,\theta) = \mathbb{E}_{q_\theta(w|D)}[\log(q_\theta(w|D)) - \log(p(w))] - \mathbb{E}_{(q_\theta(w|D)}[\log(p(D|w))]$$

We can further modify this equation by combining the two expectation terms using following formula

$$\mathbb{E}[X] + \mathbb{E}[Y] = \mathbb{E}[X+Y]$$

$$\Rightarrow F(D,\theta) = \mathbb{E}_{q_\theta(w|D)}[\log(q_\theta(w|D)) - \log(p(w)) - \log(p(D|w))]$$

The next step is to do a reparametrization trick. It has two steps, the first one being a change of variables. Let us define a variable $w^{(i)}$ which will be used as an estimation for w. Our new variable is declared as a function

$$w^{(i)} = g(\theta, \epsilon) = \mu + \sigma\epsilon, \tag{3}$$

here $\theta = (\mu, \sigma^2)$ where μ and σ^2 are the mean and variance of the variational posterior $q_\theta(w|D)$ and ϵ is sampled from a normal distribution $p(\varepsilon) \sim N(0,1)$. This gives us a way to sample values from variational posterior $q_\theta(w|D)$.

The second step in reparametrization trick is known as the Law Of The Unconscious Statistician (LOTUS). It proposes that if we know a base distribution $\epsilon \sim p(\varepsilon)$ and want to know the expectation for $w^{(i)}$, we can actually use the base distribution to calculate the expectation for $w^{(i)}$ since $w^{(i)}$ is a fucntion of ϵ. Here is a demonstrative example with more general variables. Let us say that we have a continous variable X which is defined with a distribution $p_\theta(x)$. The expected value for X is calculated in a following way

$$E_{p_\theta(x)}[X] = \int x p_\theta(x) dx$$

If we now have a new variable Y which is is defined with a distribution $p_\phi(y)$ and is also a function of variable X. The expected value for Y would be

$$Y = f(X)$$

$$E_{p_\phi(y)}[Y] = \int y p_\phi(y) dy$$

Instead of calculating the expected value for Y in this way, according to the LOTUS we can use the base distribution for calculating $E_{p_\phi(x)}[Y]$ since Y is a function of variable X.

$$\Rightarrow E_{p_\phi(x)}[Y] = E_{p_\theta(x)}[f(X)] = \int f(x) p_\theta(x) dx$$

Let us now apply this reparametrization trick into our loss function. First we sample $w^{(i)}$ using the parameters of our variational posterior $q_\theta(w|D)$ and update the loss function.

$$F(D,\theta) \approx E_{q_\theta(w^{(i)}|D)}[\log(q_\theta(w^{(i)}|D)) - \log(p(w^{(i)})) - \log(p(D|w^{(i)}))]$$

Since $w^{(i)}$ is a function of ϵ, according to the LOTUS we can calculate $E_{q_\theta(w^{(i)}|D)}$ as $E_{p(\epsilon)}$.

$$F(D,\theta) = E_{p(\epsilon)}[\log(q_\theta(w^{(i)}|D)) - \log(p(w^{(i)})) - \log(p(D|w^{(i)}))]$$

To estimate this we can use Monte Carlo sampling

$$F(D,\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \log(q_\theta(w^{(i)}|D)) - \log(p(w^{(i)})) - \log(p(D|w^{(i)})) \tag{4}$$

and finally we end up with a completely tractable equation. Using this as our loss function solves the problem we had with backpropagation. This is how Bayes by Backprop basically makes applying Bayesian learning to larger models computationally possible.

3.2.2. Local Reparametrization Trick

There is still one problem with our model. When calculating the stochastic gradients, the variance of the gradients are crucial for the performance of the process. Let us take a closer look into this problem by demonstrating the reparametrization with the log likelihood term. We can assume that our KL-divergence term from Equation 4, which is currently

$$\log(q_\theta(w^{(i)}|D)) - \log(p(w^{(i)})),$$

can be approximated with a similar reparameterization as the log likelihood term. Let us denote the log likelihood as $L_i = \log(p(D|w^{(i)}))$. By applying the Monte Carlo sampling to it we end up with $L_D = \frac{1}{N}\sum_{i=1}^{N} L_i$. Let us now calculate the variance for this term.

$$\begin{aligned}\text{Var}[L_D] &= \frac{1}{N^2}\Big(\sum_{i=1}^{N}\text{Var}[L_i] + 2\sum_{i=1}^{N}\sum_{j=i+1}^{N}\text{Cov}[L_i, L_j]\Big) \\ &= \frac{1}{N}\text{Var}[L_i] + \frac{N-1}{N}\text{Cov}[L_i, L_j] \end{aligned} \tag{5}$$

Hence with even a moderately large N the variance depends almost completely on the covariance. This can cause some problems, e.g. the gradient descent is very slow with a large variance.

The Local reparametrization trick [29] solves this problem by proposing an estimator for the variance, where $\text{Cov}[L_i, L_j] = 0$. Applying this into Equation 5 crucially changes it and we end up with a new equation

$$\text{Var}_{est}[L_D] = \frac{1}{N}\text{Var}[L_i]$$

This estimation basically scales the variance as $1/N$. In general situations, if $w^{(i)}$ is defined through some function, e.g., $f(\epsilon)$, then ϵ only influences L_D through that function. In order to make this estimation computationally even more efficient, we will not sample ϵ directly. Instead we only sample the intermediate variables $f(\epsilon)$ that influence L_D. This basically translates the global uncertainty in the weights into a local uncertainty, yielding a computationally and statistically very efficient gradient estimator.

In our BNN model, the only way the weights $w^{(i)}$ affect the expected log likelihood is through the layer activations b [29]. This allows us to utilize the Local reparametrization trick in our equation. Instead of sampling the weights $w^{(i)}$, we sample neuron activations $b = A_i W$, where A_i is the input feature map from previous layer and W is the weight matrix of our current layer. The variational posterior is translated into $q_\theta(b_j|A_i)$ and we can calculate values for activations b with a following equation [30]

$$b_j = A_i W = A_i * \mu_i + \epsilon_i \odot \sqrt{A_i^2 * \sigma^2}$$

$$\sigma^2 = \alpha_i \odot \mu_i^2,$$

where $\odot$ denotes component-wise multiplication and $*$ denotse the convolutional operation. Bayes by Backprop is still the most important thing that makes our BNN model computationally efficient, but the local reparametrization trick also helps in enhancing the performance of this model.

3.2.3. Model Weights Pruning

One of the consequences of replacing the floating-point values with probability distributions is that the number of parameters in the model is doubled. For every connection between nodes, also known as the weight, two values are calculated instead of just one: mean and variance. This makes the comparison between the CNN and BNN models a bit unfair since the BNN has a twice the number of parameters compared to the CNN. In order to make the comparison fair, we reduce the number of parameters by pruning the BNN model. By cutting the number of filters in each layer in BNN to half, the total number of parameters will be the same in both models.

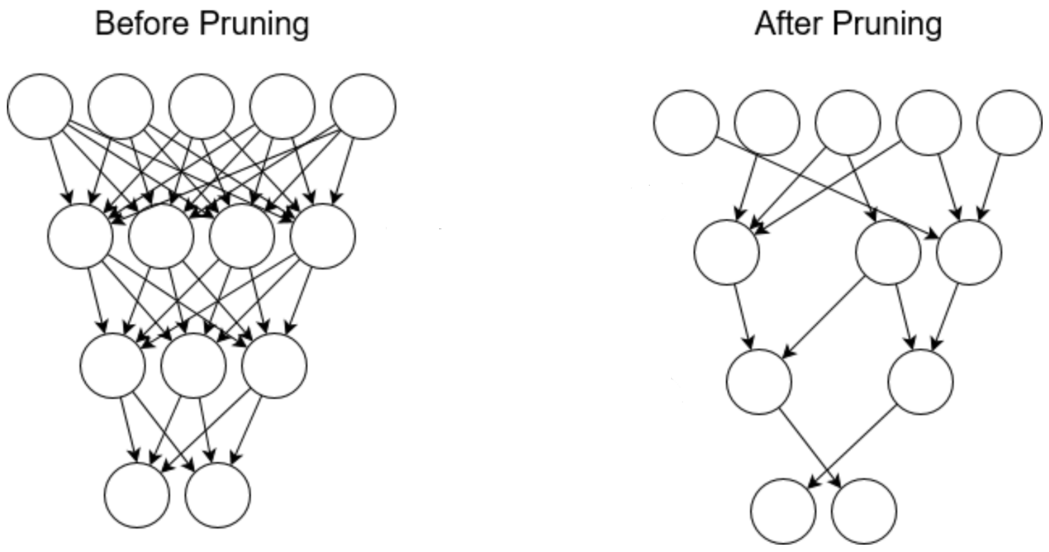

Figure 6. Illustration of model weights pruning, source [31].

In addition, both of the CNN and BNN models use model weight pruning as a way to reduce the sparsity in the networks' matrices. In other words, model pruning reduces the number of valued parameters in the network, without a major effect in the accuracy of the model. Reducing the amount of valued parameters in the network helps reduce the size of the model, thus reducing memory usage, computational cost and run-time. Model weights pruning is achieved by mapping low contributing weights to zero and reducing the number of non-zero valued weights.

Theoretically, this is done by applying a L_0 norm, which is a selector that assigns non-zero values to features that are important. Since the L_0 norm is non-differentiable, our model uses an approximation of L_0, noted by L_1. L_1 works by making a large amount of coefficients equal to zero, hence working as a regularizer and a practical feature selector for our network.[21]

4. EXPERIMENTS

This chapter will give an overview about our experiments. First, we will briefly explain our testing plan and give a short introduction to all the datasets used in the experiments. Then we will go over all the experiments and show the results from them.

4.1. Testing Plan

Our objective is to train a CNN and a BNN with three different datasets and analyze the performance of the networks from the data that is collected during the training and testing of these networks. The first dataset we consider is CIFAR10, which represents a common dataset that is used to evaluate the performances of CNNs. Our second dataset is called fruits-360, this was chosen because it can be easily edited to suit our needs and because it is a good alternative to CIFAR10, since the photos are more uniform, hence enabling better accuracy overall for the trained network. Our custom version of fruits-360 is a smaller dataset with a total of 6502 images. Our third dataset is MNIST, which contains grayscale images of handwritten numbers from 0 to 9. MNIST was chosen because with it, neural networks are known to achieve high accuracy in many different types of architechtures of neural networks, which is needed to showcase the proper uncertainty ratings of the BNN.

An epoch is a step in training a neural network, where all of the training data is fed through the network once, including adjusting the weights between the nodes with backpropagation. Our tests will consist of measuring the time it takes to complete one epoch and the whole training of the model, which is equal to 200 epochs, for both networks. During the training, the network tells its training and validation accuracies after each epoch. Training accuracy tells the accuracy of the network when using the images from the training data, and validation accuracy tells the accuracy of the network when using images from a separate set of images, that are outside of the training data. We will compare the validation accuracies over each epoch for both of the networks for CIFAR10 and our version of the fruits-360 datasets. With the fruits-360 dataset we will also measure how a decreased amount of data affects the accuracies and training time of each network. With all of these tests we aim to show the performance of each network in terms of accuracy, increase in accuracy relative to the number of epochs, and time taken for each epoch. The last experiment will only be performed on the BNN where we will train the network and see how it performs on classes it was not trained on in addition to white noise. In this separate testing we will report both the mean and standard deviation of the predictions so that we can better showcase the BNN's ability to express uncertainty.

Apart from the last experiment where we test the uncertainty ratings given by the BNN, all of our tests were run 5 times to get an average of those tests. For example, the first epoch accuracies where determined by a random value, so the networks could start at an accuracy anywhere between 0.05 and 0.25, possibly even outside of those values. This lead to us smoothing out the randomness of the results by training each network for 200 epochs, and repeating it 5 times, where we could calculate a simple arithmetic mean by getting the sum of the accuracies of each training session for each epoch, and dividing it by the amount of training sessions, which was in our case 5.

This gave us an accurate representation of how the accuracies developed throughout the training by showing us an average value for each epoch instead of a value from one single point in one of the training sessions.

4.1.1. CIFAR10

CIFAR10 is a dataset that consists of 60000 color images with 32x32 pixels that are separated into 10 classes. The dataset has classes for common animals and vehicles and each of the classes has 5000 images, the remaining 10000 images are reserved for testing. The images in CIFAR10 have more variance in terms of the shapes present in images between the classes, but also in the colours within the classes. As an example, a car is vastly different in terms of its outer shape, when compared to a cat or a dog, but also a car can be many different colors, but still be classified in the same class. This provides a good counterbalance to our more uniform fruits-360 dataset.

4.1.2. Fruits-360

Fruits-360 is a publicly available dataset from Kaggle. The dataset contains 100x100 pixel RGB images of fruits and vegetables that are separated into 131 classes. The dataset has 90483 images in total. We will utilize fruits-360 to create a custom version of it, where we have 10 classes of fruits of our choosing, each having around 500 training images and 160 testing images per class. This comes out as 6502 images in total, 4871 images for training and 1631 images for testing. Fruits-360 has a good uniformity to the images, where each class has the fruits displayed in the same size, with same photography conditions, such as lighting, background and so forth. The fruits are photographed with slight differences in the angle of the fruit, capturing the fruits shape and characteristics from every angle.

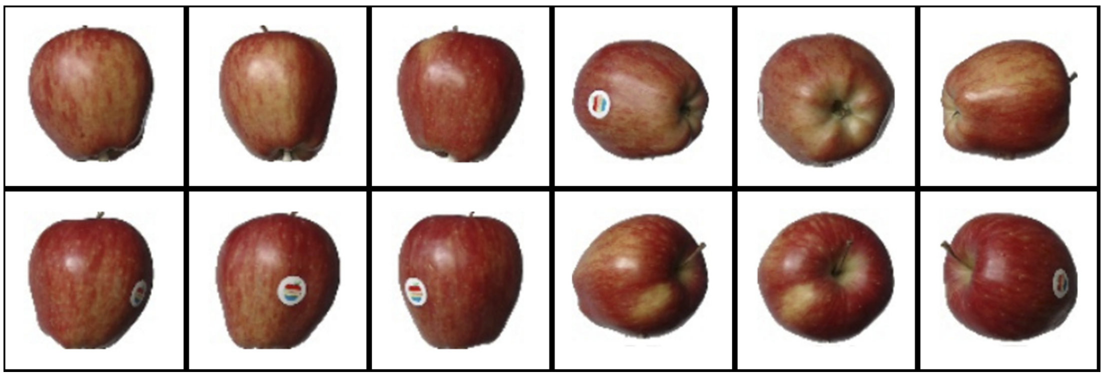
Figure 7. Example images of "Apple" class from fruits-360

4.1.3. MNIST

MNIST is one of the most common datasets used to test the accuracy of image recognition neural networks. It is a dataset of 70 000 greyscale images of handwritten digits between zero and nine. The collection contains a training set of 60 000 images, and a testing set of 10 000 images.

4.2. Performance of the Networks with CIFAR10 and Fruits-360

In this section we will go over the results we got from training our networks with CIFAR-10 and fruits-360 datasets. The main focus is on the validation and training accuracies.

Epoch	BNN	CNN
10	48.1%	55.3%
25	54.5%	60.7%
50	60.7%	64.0%
100	63.1%	65.9%
200	63.2%	66.2%

Table 1. Average validation accuracies for BNN and CNN from 5 tests with CIFAR10 dataset

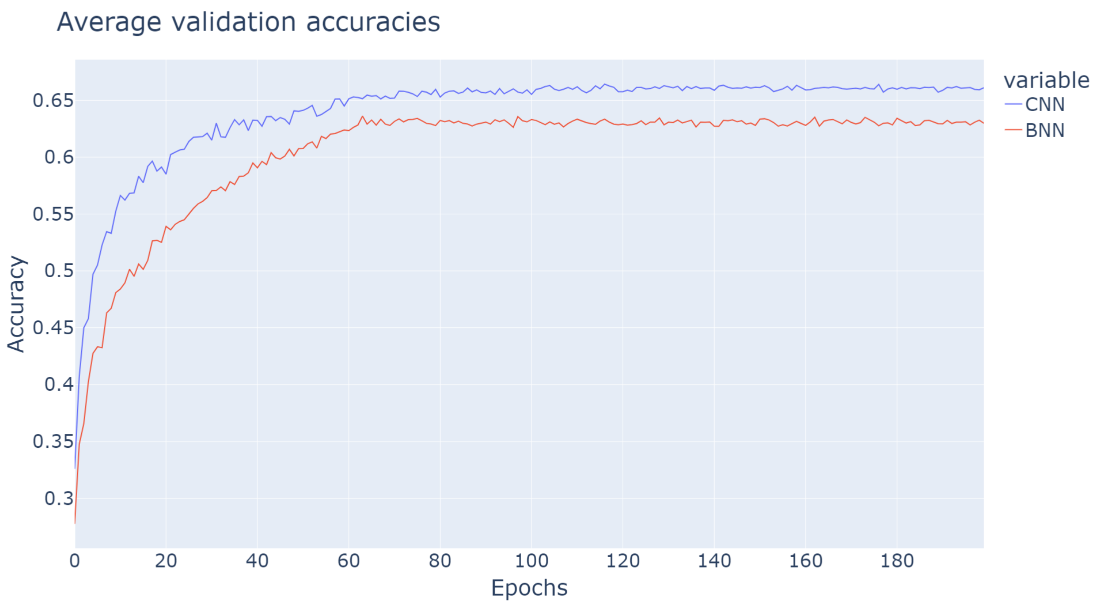

Figure 8. Average validation accuracies for BNN and CNN from 5 tests with CIFAR10 dataset

Figure 8 shows the average validation accuracies of the models when trained with the CIFAR10 dataset. The results strongly suggest that the CNN model is more accurate overall. The final validation accuracies with 200 epochs are 66% for CNN and 63% for BNN, the difference being no bigger than 3%, (see Table 1). In addition to the CNN being more accurate at the end, the accuracy also increases slightly faster in the first quarter of the epochs.

Epoch	BNN	CNN
10	48.3%	55.5%
25	57.0%	64.3%
50	67.8%	70.1%
100	75.3%	77.6%
200	75.3%	78.7%

Table 2. Average training accuracies for BNN and CNN from 5 tests with CIFAR10 dataset

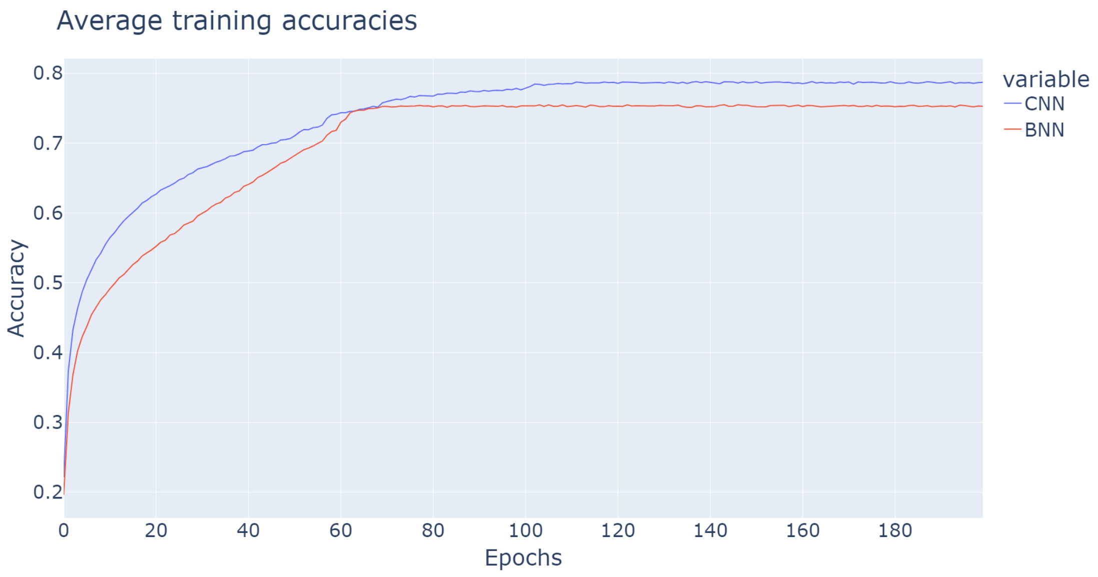

Figure 9. Average training accuracies for BNN and CNN from 5 tests with CIFAR10 dataset

Average training accuracies were also tracked for CIFAR10 and the results are shown above in Figure 9. We can see that the training accuracies with CNN and BNN behave in a same way as the validation accuracies, both models having about 12% higher training accuracy than validation accuracy. CNN got average training accuracy of 79% while BNN managed to get 75%, (see Table 2). The difference is about the same compared to the corresponding validation accuracies.

Epoch	BNN	CNN
10	69.4%	91.3%
25	75.3%	98.9%
50	81.2%	99.9%
100	91.9%	99.9%
200	94.3%	99.8%

Table 3. Average validation accuracies for BNN and CNN from 5 tests with fruits-360 dataset

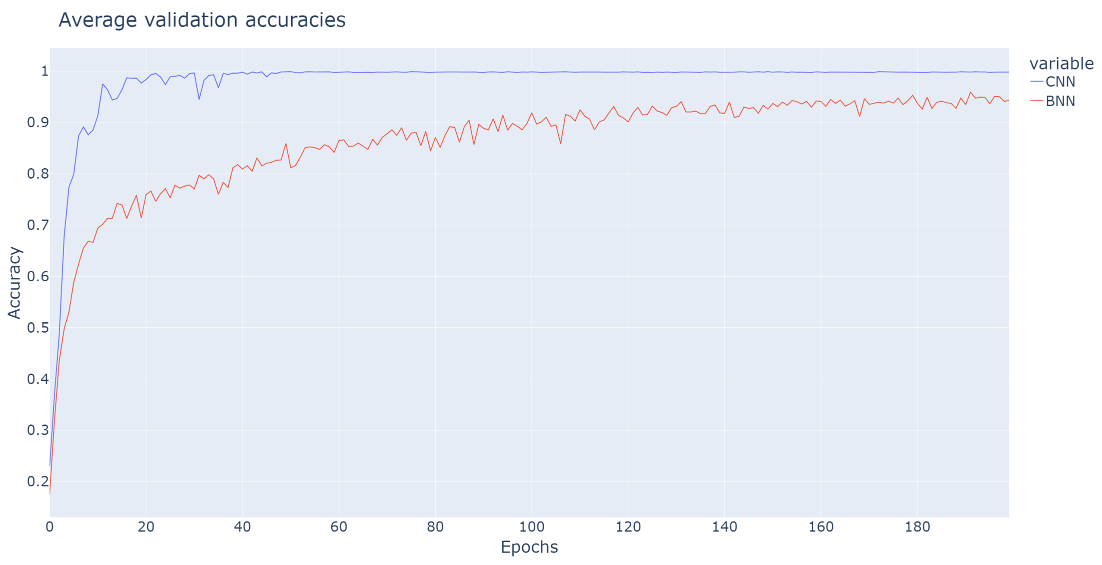

Figure 10. Average validation accuracies for BNN and CNN from 5 tests with fruits-360 dataset

Figure 10 shows the average validation accuracies of the models when trained with the fruits-360 dataset. CNN shows better accuracy overall and a higher growth rate. Similar behaviour was also found in CIFAR-10 results. With the BNN and the CNN, the initial increase in accuracy stops after the first 10 epochs or so, at this point the BNN manages to rise to an accuracy of 70%, while the CNN climbs up all the way to around 95%. Main differences between the fruits-360 and CIFAR10 datasets are the overall accuracy reached, and with the BNN, the validation accuracy does not show signs of increasing after around 70 epochs with CIFAR10, whereas with fruits-360 it seems to settle after 120 epochs to an accuracy of around 93-95%.

Epoch	BNN	CNN
10	68.5%	94.3%
25	77.8%	98.4%
50	84.5%	99.9%
100	91.0%	99.9%
200	94.7%	99.9%

Table 4. Average training accuracies for BNN and CNN from 5 tests with fruits-360 dataset

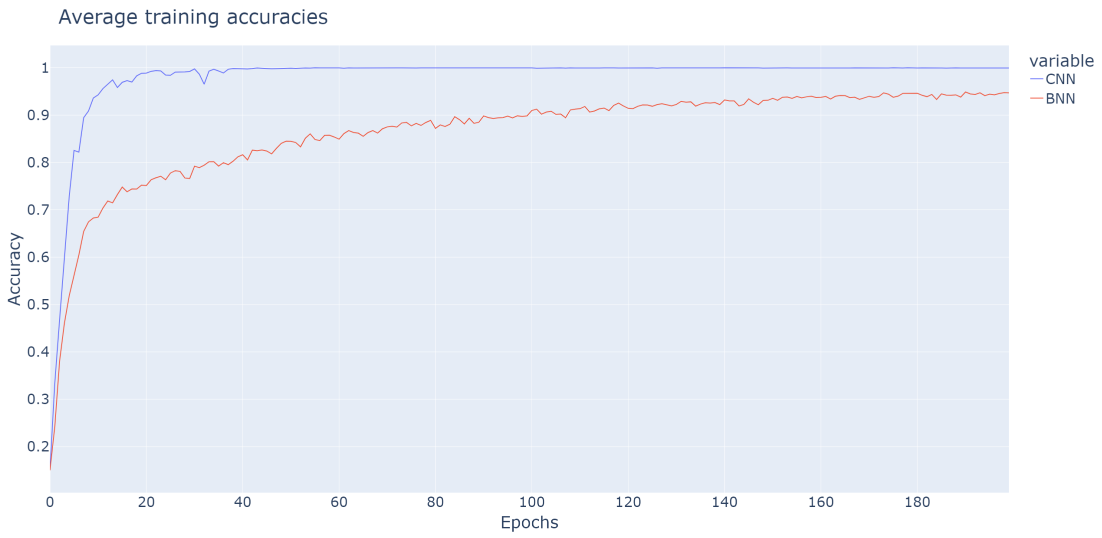

Figure 11. Average training accuracies for BNN and CNN from 5 tests with fruits-360 dataset

Unlike with CIFAR10, the training accuracies are roughly the same with both the CNN and BNN when compared to the validation accuracies, (see Table 3 and Table 4). This is due to the fruits-360 having a less diverse set of images within and across the classes than CIFAR10.

	CNN	BNN
Total time for 200 epochs	1607s	1612s
Average time per epoch	8.03s	8.06s

Table 5. Times for training with CIFAR10

	CNN	BNN
Total time for 200 epochs	5466 s	3897 s
Average time per epoch	27.33 s	19.48 s

Table 6. Times for training with CIFAR10 (CPU Only)

	CNN	BNN
Total time for 200 epochs	868 s	926 s
Average time per epoch	4.34 s	4.63 s

Table 7. Times for training with fruits-360

There is no significant difference in the training times of the models. Normally the training time of a BNN should be twice as much compared to a CNN, since the amount of parameters is doubled in a BNN, due to distributions needing to learn two values, the mean and the variance. In our case however, the BNN model is pruned down by reducing the amount of filters in each layer to half. This means that the amount of parameters is the same in both models, which causes the training times to be about the same as well. The difference in training times of CIFAR10 and fruits-360 is due to CIFAR10 being a larger dataset with more images, and also partly due to the experiments being run on different machines.

One thing to note about the training times is that the networks were trained using cuda cores from Nvidia GPUs to decrease the time it took to run the tests, but it also affected the ratios between the networks when comparing to tests that were run purely on CPUs. Table 6 shows a clear difference in the training times of the CNN and BNN with the CIFAR10 dataset. In this case the BNN is 1569 seconds faster, which is nearly a 30% difference. The efficiency of cuda cores can be noticed by the times being cut in half or less when running the tests using cuda.

4.3. Reduced Data Amount Results

We tested how the amount of data affected the learning rate and accuracy of the two different networks with our custom version of fruits-360 dataset. Since the dataset has a good uniformity within the classes, we were able to reduce the size of the dataset to 50% or 25% by retaining only every other image, or in some cases every fourth image. This ensured that the classes would contain images from all angles of the fruit despite the removal of a large percentage of the images.

Figure 10 shows the base validation accuracies over 200 epochs for the BNN and CNN. Table 7 demonstrates negligible difference in the training time of each network. With unreduced data, a CNN is faster in terms of accuracy increase for each epoch, and roughly equal in terms of CPU time for each epoch.

Epoch	BNN 50%	CNN 50%	BNN 25%	CNN 25%
10	26.5%	81.3%	17.3%	53.6%
50	8.9%	99.7%	8.8%	98.8%
100	8.5%	99.7%	10.2%	98.9%
200	9.5%	99.7%	9.8%	99.3%

Table 8. Average validation accuracies for BNN and CNN from 5 tests with fruits-360 dataset (percentages showing how much of that data is left on the dataset)

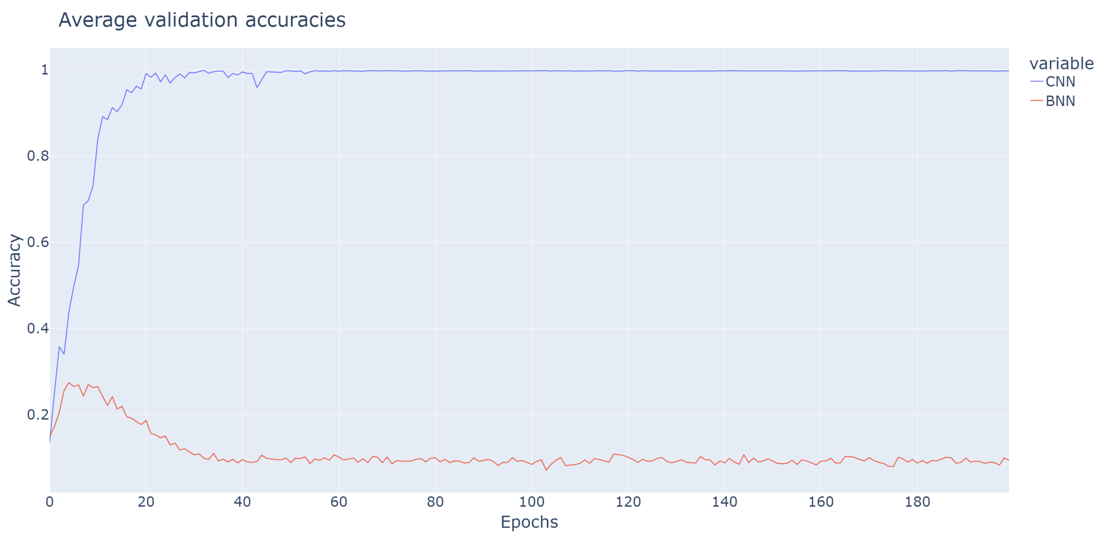

Figure 12. Average validation accuracies for BNN and CNN from 5 tests with fruits-360 dataset (50% of data)

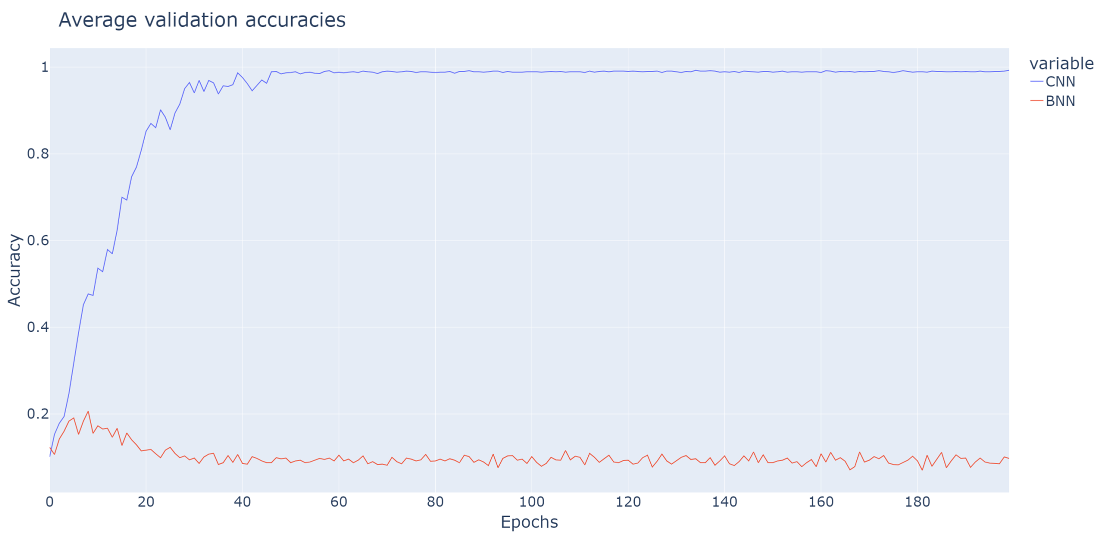

Figure 13. Average validation accuracies for BNN and CNN from 5 tests with fruits-360 dataset (25% of data)

Figures 12-13 and Table 8 show how the BNN drops off tremendously in accuracy when reducing the data by 50% and 75%, resulting in a final accuracy of around 10% after 200 epochs. In both occasions the validation accuracy starts off higher than where it settles down, which is an indicator that these smaller amounts of data are not sufficient for our BNN implementation. The CNN only shows slight accuracy decrease when the amount of data is reduced by 75%, going from a near 100% accuracy to around 98%.

Regarding the training times with the reduced data amounts, there is no noticeable difference between the BNN and CNN within the same data amounts, but the training does become slightly faster the less data there is, (see Table 9).

	CNN 50%	BNN 50%	CNN 25%	BNN 25%
Total time for 200 epochs	697.6 s	689.3 s	598.6 s	600.9 s
Time per epoch	3.58 s	3.45 s	2.99 s	3.00 s

Table 9. Times for training with reduced data

4.4. Uncertainty Estimation

We trained a Bayesian AlexNet network using the MNIST dataset. The training was done for 200 epochs using Bayes by backprop without the local reparametrization trick. We utilize uncertainty estimation methods that are specified in an article by Paras Chopra [32]. The way we estimate uncertainty is by sampling 25 separate networks that sample the weights that were learned in the training process. Due to the fact that the weights are probability distributions instead of fixed values, every time we sample the learned weights, we get slightly different values for the networks. Each of these networks make a prediction that it outputs as a logsoftmax value. We take the mean of these logsoftmax values and use the maximum value as the final prediction. The network also has the option to refuse making a prediction. This is done by taking the median of the logsoftmax values and setting a threshold for it. We will use 0.8 as the threshold for our network which means that the network needs to reach a certainty higher than 80% for a label for it to make a prediction. We train and use the same group of sampled BNNs to make prediction on three separate data sources. First is the MNIST dataset which the network was trained on. Second, we used the notMnist dataset which was designed to fool a network that was trained on the MNIST dataset. Thirdly we will generate random white noise images, which we will use as a input on the network to see what percentage of the images does the network refuse to make a prediction on.

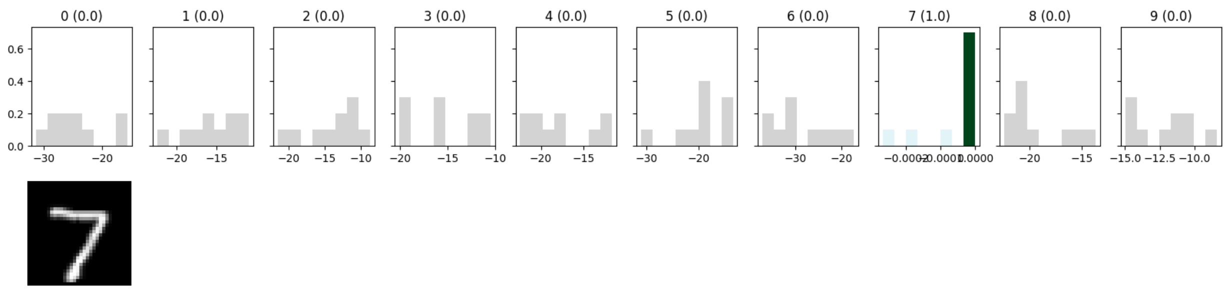

Figure 14. A successful prediction of an MNIST label

Figure 14 shows how our group of sampled BNNs outputs its prediction. Each of the labels has its own histogram and each one of the sampled networks has its own value represented. The horizontal axis of each histogram represents the ranges for the logsoftmax values and the vertical axis shows the percentage of the networks that have given a certain prediction value. For example, the histogram for label '7' has a very high column on the right-hand side of the graph. This means that the majority of the networks have decided on the value zero. One can obtain the confidence of these networks by simply exponentiating the logsoftmax value.

$$e^{LM} = C \tag{6}$$

Where LM = logsoftmax value and C = confidence. For logsoftmax value zero, equation 6 gives a confidence of one, which means that the network has a 100% confidence in its prediction. The majority of the networks have the same output value which results in an overall 100% confidence. This is indicated above the '7' histogram. The network overall achieved similar results on the rest of the data in the dataset. This was expected due to the fact that during training, the network achieved a validation accuracy of over 99% which indicates that the network is highly capable of distinguishing between the different labels in the dataset. However, as a result of the relatively high confidence threshold that we set for our network, some of the outputs were rejected by the network. If the network was forced to make a prediction on the rejected outputs, there is a high likelihood that prediction would be correct, despite the confidence not reaching the needed threshold. However, the image in Figure 15 demonstrates that it is warranted that the network expresses some uncertainty.

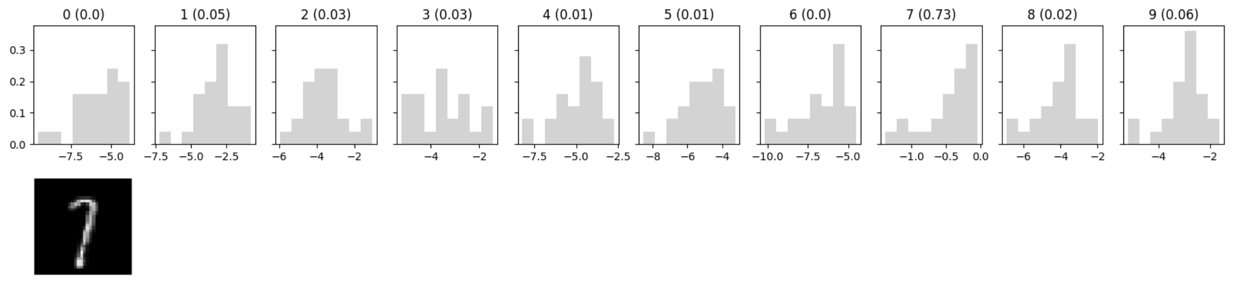

Figure 15. A example of a rejected MNIST image

In Figure 15, none of the histograms are highlighted, meaning the network did not reach the required amount of confidence to make a prediction. Overall, looking at the histograms we can see that logsoftmax values are evenly spread across. The highest confidence level of 0.73 can be found on label ’7’ which in this case would have been the correct prediction. However, looking at the rejected image we can understand why the network gave more serious consideration for labels such as ’1’ and ’9. If we wished for the network to classify this image we could either remove or lower the required confidence level. Doing this results in a smaller amount of data rejection but it will also lower the accuracy of the network due to some images being mislabeled by the network.

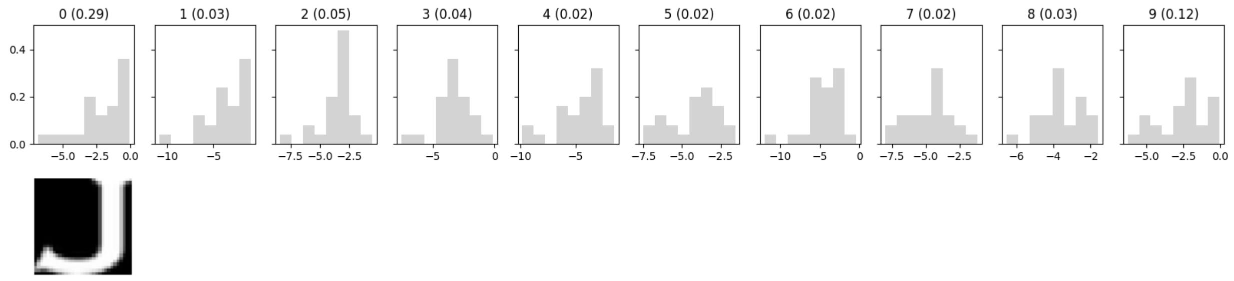

Figure 16. A example of a rejected notMNIST image

The second experiment with executed with the same group of sampled BNN networks achieved expectantly worse results. The networks managed to reject a sizable portion of the data that was fed to it. However, nearly half of the data led to predictions that exceeded the confidence threshold, which resulted in a false prediction by the network. In this experiment the network has two things working against it: the network was not trained on this dataset and this particular dataset was designed to fool networks that were trained on the MNIST dataset.

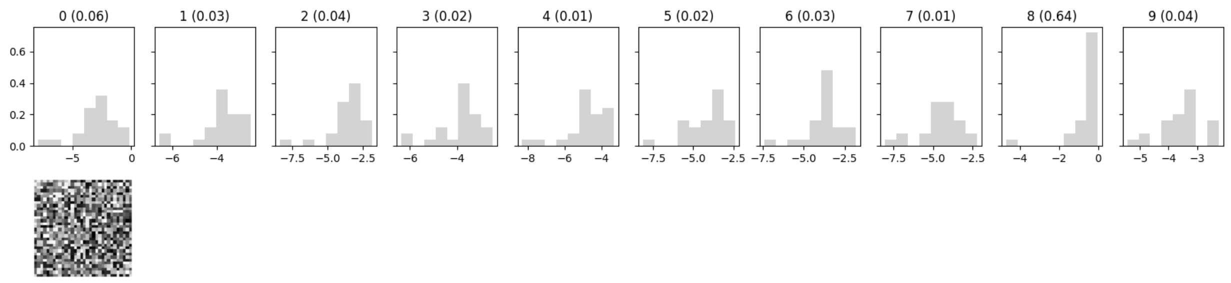

Figure 17. A example of a rejected white noise image

In comparison to the second test the third test showcases similar results. In half of the cases, the network is able to refuse making a prediction, but still a significant portion of data passes the threshold and causes the network to make a false prediction. During testing it was found that increasing the threshold affected the results of the white noise experiment significantly more than it did the notMnist experiment. In the case of the notMnist experiment, increasing the threshold resulted in a roughly 10% increase in rejected data whereas with the white noise images the number of images that the network made a prediction on was on average halved.

	MNIST	notMNIST	White noise
Total images	10000	16852	100
Predictions made	9839	8023	22
Accuracy of predictions	99%	19%	4.5%

Table 10. Overall results

4.5. Results Summary

As a summary of the results, the overall performance is favoring the CNN. The CNN had a faster learning rate in terms of increased accuracy per epoch and a better overall accuracy in both CIFAR10 and fruits-360 datasets. The experiments with reduced amounts of data showed that the BNN is even more data dependent than the CNN. The lowering of the amount of data in the dataset resulted in a complete failure in the BNN's ability to accurately approximate the true posterior, whereas the CNN was able to reach results comparable to the first experiment with the only difference being the amount of epochs it took to get to the final accuracy. The networks' times for training where comparable to each other because of the model pruning that was implemented in the BNN model. In the uncertainty estimation experiment we found that the network can express its uncertainty well when it is dealing with data that it is well familiar with. The network was trained on the MNIST dataset and it reached a validation accuracy of 99%. When testing data was fed to the network, the BNN consistently made correct predictions and refused making predictions on the images that were less clear. When the network was tested on the unfamiliar data, the results worsened significantly. Half of the notMNIST dataset and a fifth of the white noise passed the uncertainty check.

5. DISCUSSION

In this chapter we will give you a short overall reflection on our thesis work. We will discuss about some of the problems we encountered during the testing process and how we overcame them. We will also discuss how our results compared to the previous findings in related work and give a couple of ideas for possible future work.

5.1. Project Reflection

Our initial target was to compare the performances of a Bayesian neural network and a convolutional neural network. One of the original tasks which we were going to use to measure the performance of our network was emotion recognition. As the scope of the thesis became clearer we understood that we should opt for the more common performance measurement tasks. This resulted in us abandoning emotion recognition as we focused more on the comparison of the neural networks. We managed to successfully familiarize ourselves with the source code from Shridhar et al. [21] and implement our own version with our own tests. Unfortunately, we could not replicate results from the original study. We believe this is because the study used different initializations for the parameters of the experiments instead of the ones that came with the source code. Our results are closer to the results that were presented in Yi-Pei Chan's report [33] where he recreated the Bayesian LeNet from [21]. We also had some issues with performing our own experiments. The source code included an implementation of the regular version of AlexNet, whereas the experiments in [21] used the AlexNetHalf model where the number of filters is halved. Redoing the experiments when the training of the networks takes close to half an hour resulted in a lot of wasted time. Some of the limitations in showcasing uncertainty estimation with our BNN are due to the fact that not all of our datasets had enough data or the reached accuracy was not high enough to provide reliable results from the uncertainty tests. We were able to overcome this by using a different dataset, MNIST, to have the uncertainty estimation work as expected. With the initial expectations of the CNN beating the BNN, our results met our expectations. What came as a bit of a surprise was how much the BNN fell in accuracy with the smaller data amounts. It became quickly clear to us that the BNN is even more data dependent than the already famously data hungry CNN.

5.2. Literature

Our project results were in line with previous findings in related work and they do confirm their results further. The key similarities where in the training and validation accuracies being better with a CNN, and also reaching those accuracies faster with the CNN [21]. Our results with the testing of uncertainty on the BNN were also in line with the findings of related work [33][34].

5.3. Future Work

One possible path to take from here is to continue testing these models with various different datasets. Based on our study and other studies so far it seems that the CNN is overall a bit more accurate when it comes to image classification. Testing the models with datasets that they have not been tested with so far would probably further endorse this statement.

Another idea for future work is implementing Bayesian learning to different types of neural networks. The performance of the original model and Bayesian version of the model could be compared in a similar manner we have been comparing the CNN and BNN models. This could potentially demonstrate the unique features, e.g. ability to express uncertainty, of Bayesian models more or even reveal some new information about them.

Future research with the BNN could try to explore solutions to the bigger problems that plague both the CNN and the BNN. The first problem is the large amount of data that the networks require for the training process and the second problem is the long time it takes to train these networks.

6. CONCLUSIONS

Testing the performance of a Bayesian convolutional neural network and a more traditional convolutional network showed that a CNN is faster in terms of accuracy increase per epoch and more accurate overall. Our tests showed that with small amounts of data(less than 300 images per class) the BNN performs significantly worse and the decrease of data in a given dataset seems to affect a CNN less than a BNN. This is not to say a BNN does not have its place, as it was demonstrated to being able to express uncertainty, unlike a CNN. All of these findings are in line with the results in previous studies on the topic.

To conclude, a CNN still reigns as the champion in many aspects. But as stated in other literature, the strengths of BNN lie not only in its ability to refuse to give out a prediction, but also to avoid the overfitting problem of more traditional neural networks.

7. REFERENCES

[1] Friston K., Kilner J. & Harrison L. (2006) A free energy principle for the brain. Journal of Physiology-Paris 100, pp. 70–87.

[2] Aguilera M., Millidge B., Tschantz A. & Buckley C.L. (2022) How particular is the physics of the free energy principle? Physics of Life Reviews 40, pp. 24–50.

[3] Friston K. (2010) The free-energy principle: a unified brain theory? 11, pp. 127–138.

[4] Ohata W. & Tani J. (2020) Investigation of the sense of agency in social cognition, based on frameworks of predictive coding and active inference: A simulation study on multimodal imitative interaction. Frontiers in Neurorobotics 14. URL: https://doi.org/10.3389%2Ffnbot.2020.00061.

[5] Millidge B., Seth A.K. & Buckley C.L. (2021) Predictive coding: a theoretical and experimental review. CoRR abs/2107.12979, pp. 39–43.

[6] Vandenbos G. (2015) APA Dictionary of Psychology. URL: https://archive.org/stream/APADictionaryOfPsychology2ndEdition2015/APA.

[7] Friston K. (2003) Learning and inference in the brain. Neural Networks 16, pp. 1325–1352.

[8] Jospin L.V., Buntine W.L., Boussaïd F., Laga H. & Bennamoun M. (2020) Hands-on bayesian neural networks - a tutorial for deep learning users. CoRR 17, pp. 29–48.

[9] S. Albawi T.A.M. & Al-Zawi. S. (2017), Understanding of a convolutional neural network. 2017 International Conference on Engineering and Technology (ICET). URL: https://ieeexplore.ieee.org/document/8308186.

[10] Hernández-Lobato J.M. & Adams R.P., Probabilistic backpropagation for scalable learning of bayesian neural networks. URL: https://arxiv.org/abs/1502.05336.

[11] Graves A. (2011) Practical variational inference for neural networks. In: J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira & K. Weinberger (eds.) Advances in Neural Information Processing Systems, vol. 24, Curran Associates, Inc., vol. 24. URL: https://proceedings.neurips.cc/paper/2011/file/7eb3c8be3d411e8ebfab08eba5f49632-Paper.pdf.

[12] Blundell C., Cornebise J., Kavukcuoglu K. & Wierstra D. (2015), Weight uncertainty in neural networks.

[13] Millidge B. (2021), Applications of the free energy principle to machine learning and neuroscience.

[14] Costa L.D., Parr T., Sajid N., Veselic S., Neacsu V. & Friston K. (2020) Active inference on discrete state-spaces: A synthesis. Journal of Mathematical Psychology 99. URL: `https://doi.org/10.1016%2Fj.jmp.2020.102447`.

[15] Friston K., Rigoli F., Ognibene D., Mathys C., FitzGerald T. & Pezzulo G. (2015) Active inference and epistemic value. Cognitive neuroscience 6(4), pp. 187–214.

[16] Lotter W., Kreiman G. & Cox D. (2017), Deep predictive coding networks for video prediction and unsupervised learning.

[17] Ranzato M., Szlam A., Bruna J., Mathieu M., Collobert R. & Chopra S. (2014), Video (language) modeling: a baseline for generative models of natural videos. URL: `https://arxiv.org/abs/1412.6604`.

[18] Rao R.P.N. & Ballard D.H. (1999) Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. Nature Neuroscience, Volume 2 , pp. 79–87.

[19] Valpola H. (2015), Chapter 8 - from neural pca to deep unsupervised learning.

[20] L. Jospin W. Buntine F.B.H.L.M.B. (2020) Hands-on Bayesian Neural Networks – A Tutorial for Deep Learning Users.

[21] Shridhar K., Laumann F. & Liwicki M. (2019), A comprehensive guide to bayesian convolutional neural network with variational inference.

[22] Graves A. (2011) Practical variational inference for neural networks. In: Advances in Neural Information Processing Systems, vol. 24, Curran Associates, Inc., vol. 24.

[23] Goodfellow I., Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S., Courville A. & Bengio Y. (2014) Generative adversarial nets. In: Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence & K. Weinberger (eds.) Advances in Neural Information Processing Systems, vol. 27, Curran Associates, Inc., vol. 27.

[24] Radford A., Metz L. & Chintala S. (2015), Unsupervised representation learning with deep convolutional generative adversarial networks. URL: `https://arxiv.org/abs/1511.06434`.

[25] Khan M.E., Nielsen D., Tangkaratt V., Lin W., Gal Y. & Srivastava A. (2018) Fast and scalable bayesian deep learning by weight-perturbation in adam. In: ICML.

[26] O'Shea K. & Nash R. (2015) An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458 URL: `https://arxiv.org/abs/1511.08458`.

[27] He K., Zhang X., Ren S. & Sun J. (2016) Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), vol. 1, vol. 1, pp. 770–778.

[28] Krizhevsky A., Sutskever I. & Hinton G.E. (2012) Imagenet classification with deep convolutional neural networks. In: F. Pereira, C.J.C. Burges, L. Bottou & K.Q. Weinberger (eds.) Advances in Neural Information Processing Systems, vol. 25, Curran Associates, Inc., vol. 25. URL: `https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

[29] Kingma D.P., Salimans T. & Welling M. (2015), Variational dropout and the local reparameterization trick.

[30] Neklyudov K., Molchanov D., Ashukha A. & Vetrov D. (2019) Variance networks: When expectation does not meet your expectations. In: International Conference on Learning Representations. URL: `https://openreview.net/forum?id=B1GAUs0cKQ`.

[31] Gordon M.A. (2020), Do we really need model compression?

[32] Chopra P. (2019), Making your neural network say "i don't know"-bayesian nns using pyro and pytorch. URL: `https://towardsdatascience.com/making-your-neural-network-say-i-dont-know-bayesian-nns-using-pyro-and-pytorch-b1c24e6ab8cd`.

[33] Chan Y.P. Bayesian convolutional neural networks with variational inference URL: `https://arxiv.org/abs/1511.08458`.

[34] Osawa K., Swaroop S., Khan M.E.E., Jain A., Eschenhagen R., Turner R.E. & Yokota R. Practical deep learning with bayesian principles. Advances in neural information processing systems 32, pp. 4289–4301.

[35] Kingma D.P. & Ba J. (2014), Adam: A method for stochastic optimization. URL: `https://arxiv.org/abs/1412.6980`.

8. APPENDIX

8.1. BNN Variables

Variable	Value
Number of epochs	200
Initial learning rate	0.001
Activation function type	Softplus
Data batch size	256
Validation portion	0.2
Optimizer	Adam [35]
β - type	Standard

8.2. CNN Variables

Variable	Value
Number of epochs	200
Initial learning rate	0.001
Activation function type	Softplus
Data batch size	256
Validation portion	0.2
Optimizer	Adam [35]

8.3. Source Code

The source code for the models used in this thesis were provided by Shridhar, Kumar and Laumann, Felix and Liwicki, Marcus [21]. The source code for the models is available in https://github.com/kumar-shridhar/PyTorch-BayesianCNN. The source code for the uncertainty estimation was provided by Paras Chopra [32]. The source code for the uncertainty estimation is available in https://github.com/paraschopra/bayesian-neural-network-mnist

8.4. CNN Architecture

Layer type	width	stride	padding	Input shape	nonlinearity
convolution (11 x 11)	64	4	5	M x 3 x 32 x 32	Softplus
max-pooling (2 x 2)		2	0	M x 64 x 32 x 32	
convolution (5 x 5)	192	1	2	M x 64 x 15 x 15	Softplus
max-pooling (2 x 2)		2	0	M x 192 x 15 x 15	
convolution (3 x 3)	384	1	1	M x 192 x 7 x 7	Softplus
convolution (3 x 3)	256	1	1	M x 384 x 7 x 7	Softplus
convolution (3 x 3)	128	1	1	M x 256 x 7 x 7	Softplus
max-pooling (2 x 2)		2	0	M x 128 x 7 x 7	
fully-connected	128			M x 128	

8.5. BNN Architecture

Layer type	width	stride	padding	Input shape	nonlinearity
convolution (11 x 11)	32	4	5	M x 3 x 32 x 32	Softplus
max-pooling (2 x 2)		2	0	M x 32 x 32 x 32	
convolution (5 x 5)	96	1	2	M x 32 x 15 x 15	Softplus
max-pooling (2 x 2)		2	0	M x 96 x 15 x 15	
convolution (3 x 3)	192	1	1	M x 96 x 7 x 7	Softplus
convolution (3 x 3)	128	1	1	M x 192 x 7 x 7	Softplus
convolution (3 x 3)	64	1	1	M x 128 x 7 x 7	Softplus
max-pooling (2 x 2)		2	0	M x 64 x 7 x 7	
fully-connected	64			M x 64	

8.6. Contributions Of Each Participant

The contributions from each participants were shared fairly evenly throughout the thesis. Each member of this thesis wrote a section or two to chapter 1. Each of us also contributed in researching the related work and explaining the contents of them in chapter 2. In chapter 3 Paananen handled the Convolutional neural network section, Antikainen handled the Bayesian neural network and Model weights pruning sections and Tölli handled Bayes by Backprop and Local Reparametrization trick sections. The responsibilities in chapter 4 were shared so that Antikainen handled all the testing for Fruits-360 dataset, Tölli handled testing related to the CIFAR10 dataset and Paananen did the uncertainty estimation with the MNIST dataset. Some modifications to the source code was done by each member of the thesis. Each of us wrote down the results and conclusions from the experiments that one was in charge of. Final chapters 5, 6, 7 and 8 were results from all of our contributions.