NTNU
Kunnskap for en bedre verden

Department of Computer Science

TDT4173 - Assignment 3

Introducing Probabilistic Learning to a Feed Forward Neural Network

Group:
Group 14

Authors:
Jonatan Engstad (jonatane)
Adrian Langseth (adrianwl)
Miriam Woldseth (miriamvw)

July 14, 2021

Abstract

The problem of classifying objects into predefined classes is a common decision making task within many fields, like medicine, science, and business. For some of these classification tasks, the ability to show uncertainty can be beneficial, for example in order to identify instances that should be further examined. A possible solution for automating such tasks are to utilize probabilistic methods. In this paper two probabilistic approaches are integrated into a simple feed forward neural networks (FFNN) in order to explore how the addition of these methods affect the network. These introduced methods result in two additional networks, namely a FFNN with dropout that works as a Bayesian approximator, and a Bayesian neural network (BNN) which is a Bayesian version of the simple FFNN. These methods are tasked with classifying handwritten digits from the MNIST data set, and are compared on their precision, recall, F1-score, and accuracy. Furthermore, their ability to show uncertainty is explored by running the networks on a test set of anomalies from the notMNIST data set of letters and evaluated based on the entropy of their confidence in their predictions at the output. Both the accuracy and ability to show uncertainty of the networks are investigated for different sizes of the training set.

The results show that both of the FFNNs have better precision, recall, F1-score, and accuracy than the BNN when the networks are trained on 50000 instances with a validation set of 10000 instances. However, the accuracy of the BNN is better for a training set size of 1000 instances. Furthermore, the BNN has higher entropy on both the train, test, and anomaly set than the FFNNs, and is thus better at showing uncertainty when presented with anomalies and difficult queries. The paper thus concludes that BNNs can be more beneficial for classification tasks were the ability to show uncertainty is essential.

Repository for project: `https://github.com/AdrianLangseth/TDT4173_Anomaly_Detection/tree/main`

Web page: `https://mirree.github.io/`

1 Introduction

The classification of objects into a predefined group or class is a frequently encountered decision-making task for humans [Zhang, 2000, Saucedo, 2007]. Humans can classify objects based on the observed attributes and characteristics of the objects, thus solving a classification problem. Classification problems are common within several fields, including business, science and medicine, and the frequent need for solving classification problems indicates that automation of such tasks may be advantageous. There are several ways to build and train systems to solve classification problems. This paper will focus on how neural networks can be used to solve such problems.

Neural networks have several beneficial properties, including non-linearity and adaptiveness [Zhang, 2000, Du and Swamy, 2006]. Their non-linearity makes them able to approximate any function with arbitrary accuracy, meaning they are universal approximators, making them flexible in modeling real world relationships. However, several types of neural networks require large amounts of data for training and are prone to overfitting when given little training data [Srivastava et al., 2014].

Overfitting occurs when the generalization accuracy of a neural network, the accuracy it fits examples beyond the training data with, goes from decreasing to increasing while the accuracy over the training set continues to decrease [Mitchell, 1997, p. 108-111]. There are several methods one can employ to avoid overfitting when training feed forward neural networks (FFNNs), for example Early Stopping and Dropout [Srivastava et al., 2014]. However, none of these will alone identify uncertainty in the model. The ability to identify how certain the model is with its prediction can be beneficial in several circumstances. Take the example of a network that has been trained to interpret images from screening tests for cancer and classify these. In this case it is crucial that the neural network does not make overconfident decisions, as classifying a sick person as healthy can have dire consequences. However, a neural network that is not trained to show uncertainty, like feed forward neural networks, would focus mainly on one class, i.e., the outputs of the neural network would be close to [0,...,0,1,0,...,0]. Thus, the neural network would be overconfident in its predictions, possibly wrongly classifying an image. Fortunately, there exist neural networks that can identify and communicate the uncertainty in their model, e.g., Bayesian neural networks (BNNs). BNNs have several benefits, including robustness against overfitting, working well on small data sets, and providing uncertainty estimation and probability distributions.

It is well known that BNNs are more computationally expensive than FFNNs. Gal and Ghahramani [2016] argue why representing the uncertainty in a neural network model, i.e., the model uncertainty, is important, and shows how dropout can be used as a Bayesian approximation to accomplish this. Furthermore, they argue that by using a probabilistic interpretation of dropout instead of BNNs, one can mitigate the problem of representing model uncertainty without sacrificing either computational overhead or test accuracy.

In this paper a simple FFNN, a FFNN with dropout, and a BNN are trained to classify handwritten digits from the MNIST [LeCun et al., 1998] data set. The networks are compared on accuracy, precision, recall, and F1-score in order to explore how the introduction of probabilistic learning methods to a neural network affects its performance. Furthermore, the paper explores the ability of the networks to show uncertainty when anomalies are introduced. This investigation is conducted by running a test set consisting of anomalies on all the networks. Furthermore, the different networks are trained on training sets of varying sizes in order to explore how their behavior varies with the size of the training set, and the notMNIST [Bulatov, 2011] data set of letters provides data used as anomalies in the testing set when exploring the ability of a model to show uncertainty.

In order to evaluate the uncertainty shown by a network, the entropy of the probabilities at the output is evaluated.

The goal of the project is not to implement or to compete with state-of-the-art solutions like CNNs [LeCun et al., 1998]. Rather, the goal is to investigate how the introduction of a probabilistic learning method like Bayes by Backprop can affect network performance. The networks examined in this project thus build on a minimal FFNN with only one hidden layer.

2 Data

As presented in the introduction, data from the MNIST database will be used for training and testing the models, while data from the notMNIST data set will be used as anomalies when testing the ability of the networks to show uncertainty. A short overview of the data sets are shown in Table 1.

	Size	**Source**	**Usage**
MNIST	70 000	LeCun et al. [n.d.]	Training set: 1 000 to 50 000 instances Validation set: 10 000 instances Test set: 10 000 instances
notMNIST	≈ 519 000	Bulatov [n.d.]	Anomaly test set: 10 000 instances

Table 1: Overview of the MNIST and notMNIST databases

2.1 MNIST

MNIST is a database of images of handwritten digits, and was constructed based on the NIST Special Databases 1 and 3 (respectively abbreviated as SD-1 and SD-3) [LeCun et al., 1998]. The database consists of a training set with 60 000 images and a test set with 10 000 instances, where the patterns in the training set consists of a set of approximately 250 writers that are disjoint with the set of writers used in the test set. The database was constructed by taking a subset of 35 000 images from both SD-1 and SD-3, where 30 000 of the patterns from each were collected in the training set, and 5 000 patterns from each were collected in the test set. The reasoning behind mixing these two data sets, was that SD-3 contained patterns that were much cleaner and easier to recognize than SD-1.

When constructing the MNIST database, the original NIST images were first size-normalized to fit in a 20x20 pixel box, before being centered based on their computed center of mass of pixels in a 28x28 field.

The images in the MNIST dataset are stored in an 8-bit grayscale format. To make our models able to easily handle grayscale images of other formats, each pixel had its value compressed (normalized) into the range 0-1 (inclusive), with values of 255 becoming 1 and 0 staying at zero. This is also a common technique when dealing with datasets of where features have varying ranges (to prevent certain features from disproportionately affecting predictions), something that does not occur with image data. As images are two-dimensional and our densely connected neural network layers are one-dimensional, all image data was flattened into one-dimensional arrays before being given to the network as input.

2.2 notMNIST

notMNIST is a data set of glyphs extracted from different fonts of the letters A through J [Bulatov, 2011]. The data set consists of two sets, a small, cleaned set of approximately 19 000 instances, and a large, uncleaned set of approximately 500 000 instances. Each of the two data sets have a label error rate of around 0.5% and 6.5% respectively, i.e., approximately 0.5% and 6.5% of the images are labeled incorrect. However, since the notMNIST data will only be used as anomalies, the classes are not of interest, and thus the label errors will not affect this project.

The images of the notMNIST dataset are grayscale images of the same size (28x28 pixels) as those in the MNIST dataset. This means that the only pre-processing that needs to be performed on the notMNIST dataset is normalizing each pixel value into the range 0-1, such as with the MNIST data. The instances can then be passed directly on to the models in the same fashion as the MNIST images.

2.3 Usage

As mentioned in Section 2.1, the MNIST database consists of a training set of 60 000 instances and a test set of 10 000 instances. In this project, the test set of 10 000 instances was used as is, while the MNIST training set was further split into a validation set consisting of 10 000 instances and a training set consisting of the remaining 50 000 instances. Several runs were conducted with different size training sets, with the number of instances varying between 1 000 and 50 000, increasing exponentially (i.e., the number of instances used were 1 000, 2 500, 7 000, 19 000 and 50 000). Furthermore, 10 000 instances of the notMNIST data set was used as the test set of anomalies when evaluating the ability of a network to show uncertainty.

3 Methods

This section describes the different methods explored in this project.

3.1 FFNN

In the project, the introduction of probabilistic properties to a neural network will be investigated. To accurately portray the impacts of the introduction, a baseline must be established. Without a comparative model to whom no changes are made, one cannot accurately discuss whether the implemented changes had significant effects in either direction. Therefore, the FFNN is created with the same objective as the other models, to accurately classify the MNIST images. Although a CNN would perform better [LeCun et al., 1995], the baseline performance is not the subject of the project, but rather the Bayesian introduction.

The FFNN is a very simple and rudimentary neural network. The network has a one-way information flow and does not allow cycles. This results in a conceptually simplistic network. The version implemented in the project is a two-layer model, meaning there is a single hidden layer of perceptrons between the input and the output layer. The hidden layer is the connection between the inputs and outputs, and together with the output layer tries to exhibit the underlying behavior of the objective function. This is primarily done through adaptation of weights during training, but is heavily influenced by the choice of objective function. The objective function in the implemented FFNN is the categorical cross entropy. The function is well suited for classification tasks where the classification is true for one class and false for all others, such as in this project. The function is shown in Equation (1), the function punishes harder on a logarithmic scale, with harsher penalties on confident false predictions. Specifically, the loss is calculated by the natural logarithm of the confidence y_i in the classification i multiplied by the target value, which would be 1 for the target class and 0 for all others. This means if a model is sure in its prediction of a wrong class, and leaving a low confidence in the actually correct classification, this will be heavily punished by the negative natural logarithm of its confidence in the actually correct class, incurring an infinite loss when the belief in the correct approaches zero. In the initial aim of building a model which can accurately predict MNIST images, this is a suitable function.

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\ \text{size}}} y_i \cdot \ln \hat{y}_i \tag{1}$$

3.1.1 Dropout

The FFNN was implemented twice, once as the rudimentary version of the previously described, and once with a dropout layer on the hidden layer. A dropout layer is a per-layer regularization of the nodes, which probabilistically drops the nodes, setting the outputs to 0. The probabilistic nature of the node dropping is useful in attempts to force the network to learn a robust function rather than creating single strong connections from input to output. There was not placed a dropout layer on the visible flattened layer, due to how the network perceives a dropped node. The network sees a dropped node as a non-activation, meaning we have a value of 0, leading to a conflict as it is also how the background color, black, is encoded. This entails that in the visible flattened layer, a dropped node would be virtually indistinguishable from a background pixel, making the use of a dropout layer on the visible layer counterproductive and unsuitable for the task at hand.

Dropout layers are usually only active during training, and do not take part in the testing of the network. However, following Gal and Ghahramani [2016] in the construction of the model, the dropout layer here is also active in testing. Gal and Ghahramani [2016] states that predictive probabilities output, such as the softmax output employed in the FFNN model, are often erroneously interpreted as model confidence. They argue that using dropout layers during testing introduces a certain Bayesian approximation to the model, allowing the model to better show uncertainty. During testing with the dropout layer, the stochastic aspect to the model will alter the output of the model for each prediction. Therefore, by performing prediction on the same input several times, we in effect create an ensemble of models, which produces a probability distribution of each classification. In this implementation, the class of a single image is predicted 100 times. This is a sufficiently large amount to capture the uncertainty in the model, whilst not exceeding the capabilities of the hardware used for this project. To illustrate, 100 predictions on each of the 10000 notMNIST images, for each of the 5 dropout models yields a total of $5 \cdot 10^6$ total predictions on the notMNIST dataset.

The dropout layer was used to investigate whether introducing probabilistic dropping of nodes improves the networks ability to show uncertainty when given data from a separate previously-not-seen dataset, such as in this project where the notMNIST dataset is introduced to the MNIST-trained models. The FFNN with dropout is relevant to investigate for the task at hand due to its nature as a Bayesian approximation. As the regular FFNN will be compared to the ability of the BNN to identify uncertainties, it is natural to investigate a feature to the FFNN which may apply the ability to show uncertainty such as the BNN. The dropout-modified FFNN will therefore be computationally more expensive than the regular one, however will approximate Bayesian behavior. In comparison to the BNN, the dropout model is significantly cheaper computationally and conceptually much simpler, although only an approximation to Bayesian behavior, whilst the BNN is itself a probabilistic model.

3.1.2 Hyperparameters

Hyperparameter optimization is a minimization problem of a loss function on a validation set, with the hyperparameters as variables. Using a certain search algorithm, the optimization process has the aim of finding the optimal combination of hyperparameter selections to minimize the loss function. As the initial aim in constructing the model is to accurately predict the classification of images, the loss function is set to be the negative accuracy. Therefore, the algorithm will seek to find a combination of hyperparameter selections to attain the lowest negative accuracy, and therefore also the highest accuracy.

For the use of search algorithm for the parameters, Grid Search was primarily considered because of its simplicity. However, Grid Search suffers from the curse of dimensionality, and in this case of 6 separate hyperparameters, all with a significant scope, Grid Search was too inefficient. Therefore, the Tree-structured Parzen Estimator, was used. Tree-structured Parzen Estimator is a Sequential Model-Based Optimization algorithm which uses Bayesian reasoning to select which hyperparameter combinations to investigate next. The algorithm performs well with high dimensionality and small fitness evaluation budgets[Bergstra et al., 2013]. Although the latter is not as much of an issue with the current data set, the former is. The hyperparameters which are tuned are of a high dimensionality, where several of the dimensions are continuous ranges of possibilities and not discrete choices, giving a large amount of possible combinations. This made both an exhaustive grid search extremely inefficient, and made a Random Search too inaccurate.

The relevant hyperparameters to optimize in the models were the learning rate, the optimizer algorithm, the activation function on the hidden layer and the batch size. In addition to this, the dropout rate used on half the models was optimized. The amount of epochs was not a topic in hyperparameter optimization.It was rather set to be as long as possible before resulting in overfitting. This was implemented by using Early Stopping regularization and a seemingly way too large amount of epochs. Although a large amount of epochs would intuitively lead to overfitting, Early Stopping adds a callback on each epoch of training. The callback lets the Early Stopping terminate the training if the loss on the validation set has not been improved upon over a set amount of epochs.

3.2 BNN

An issue facing neural networks is that they are trained not to recognize input, but to give a correct answer for some given input. While this difference may seem like a minor detail, its consequence is that these networks are unable to say that they do not recognize input data. This can for example lead to images that are clearly just noise being classified as some specific class with the network showing very high certainty [Nguyen et al., 2014]. BNNs are one possible solution to this problem.

A BNN is a variation on the standard (feed forward) neural network that uses random variables for its weights and biases, instead of scalar values. In practical terms this means that the network's weights and biases are assigned probability distributions instead of scalar values. Figure 1 shows a simple model of how this could look. In effect, a BNN is a (possibly infinite) ensemble of networks, with each network having its weights and biases drawn from the shared set of probability distributions that is the BNN's parameters [Blundell et al., 2015].

These attributes of the BNN method give it three main advantages compared to the standard FFNN: First, the network is able to show (un)certainty: By comparing the response of each sample network, one can get an idea of how certain the BNN is of its result. The congruence of the sample network outputs can be viewed as the networks overall certainty of its result. Second, overfitting is avoided: Due to the uncertain nature of the network, the network naturally tends to avoid overfitting. Third, the network show improved performance with smaller datasets: Due to its ability to generalize across data using uncertainty inherent to the data, BNNs avoid much of the overfitting that FFNNs usually experience when training on smaller datasets.

On the other hand, due to its ability to internalize the uncertainty of the data given to it, a BNN may suffer reduced accuracy compared to a conventional FFNN of the same structure. A careful selection of hyperparameters and network structure can amend much of this difference though, as shown by Blundell et al. [2015].

The network parameters are optimized by means of a backpropagation method for Bayesian models first described in Blundell et al. [2015] and which will be discussed in the following sections. The objective function used is the evidence lower bound (ELBO), which is the negative value of the loss function described by Equation (5).

3.2.1 Variational Inference

Blundell et al. describe how this is achieved mathematically. By viewing the neural network as a probabilistic model $P(y|x, w)$ with input $x \in \mathbb{R}^d$ and a possible output y from a set of classes Υ in the case of classification, or from $\mathbb{R}$ in the case of regression. The likelihood, $P(D|w)$ (D being the training samples) can be used to find the weights w using maximum likelihood estimation (MLE):

$$w^{\mathrm{MLE}} = \arg\max_w P(D|w) = \arg\max_w \sum_i \log P(y_i|x_i, w) \tag{2}$$

The MLE calculation is usually achieved using gradient descent. By instead optimizing for the posterior $P(w|D)$ by calculating the maximum a posteriori (MAP) of the weights w^{MAP}, regularization can be introduced:

$$w^{\mathrm{MAP}} = \arg\max_w P(w|D) = \arg\max_w \log P(w|D) + \log P(w) \tag{3}$$

Calculating the entire posterior $P(w|D)$ for a network of any proper size however, is intractable. This is because calculating the posterior requires us also calculate the evidence $P(D)$, something which requires integration over a very high-dimensional space, and which has no analytical solution [Mackay, 1995]. Blundell et al. got around this limitation however, using variational inference.

Variational learning, using variational inference, constructs an approximation of the posterior distribution $P(w|D)$, $q(w|\theta)$ and then finding a θ such that the Kullback-Leibler divergence of the two distributions is minimized - meaning that the two distributions are as similar as possible. This results in the following cost function:

$$\begin{aligned}\theta^* &= \arg\min_\theta \mathrm{KL}[q(w|\theta)||P(w|D)] \\ &= \arg\min q(w|\theta) \log \frac{q(w|\theta)}{P(w)P(D|w)} \mathrm{d}w \\ &= \arg\min_\theta \mathrm{KL}[q(w|\theta)||P(w)] - \mathbb{E}_{q(w|\theta)}[\log P(D|w)]\end{aligned} \tag{4}$$

Which can be simplified to the following formula[1]:

$$F(D, \theta) = \mathrm{KL}[q(w|\theta)||P(w)] - \mathbb{E}_{q(w|\theta)}[\log P(D|w)] \tag{5}$$

Equation (5), known as the variational free energy, has been utilized for some time, starting in statistical physics [Jordan et al., 1999]. Blundell et al. however, showed were able to derive the following, much cheaper, approximation of the cost function:

$$F(D, \theta) \approx \sum_{i=1}^{n} \log q(w^{(i)}|\theta) - \log P(w^{(i)}) - \log P(D|w^{(i)}) \tag{6}$$

$w^{(i)}$ refers to the ith Monte Carlo samples, as Monte Carlo sampling to evaluate the expectations.

3.2.2 Bayes by Backprop

Using the Equation (6), one can perform backpropagation on the Bayesian model, termed "Bayes by backprop" by Blundell et al.

During FFNN backpropagation, the gradient is computed with respect to every feature. Bayes by backprop on the other hand, computes its gradients with respect to the mean and the standard deviation. As it turns out though, this gradient is not much different from the standard backpropagation method; both the gradient of the mean and the gradient of the standard deviation parameter include in their calculations the usual gradient found by backpropagation [Blundell et al., 2015].

[1] $-F(D, \theta)$ is known as the Evidence Lower Bound (ELBO).

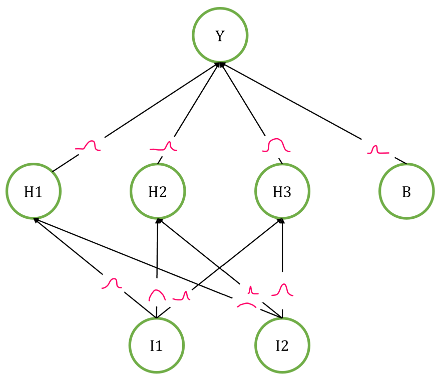


Figure 1: Each weight in the network is assigned a distrubution instead of a scalar value, as is common in standard neural networks.

Source: Adapted from Blundell et al. [2015]

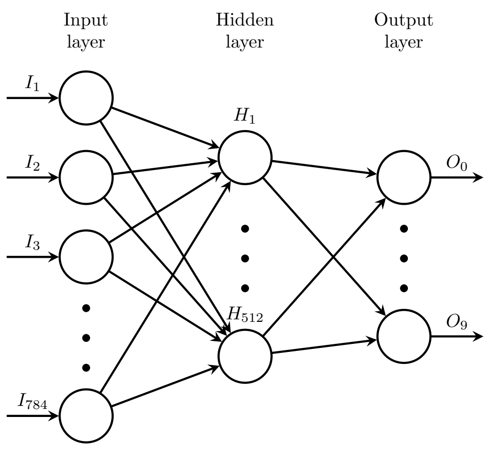


Figure 2: Simple Feed Forward Neural Network for MNIST classification. A 28×28 input dimension connected to a hidden layer of 512 nodes, leading to the 10 output nodes, each representing a classification(O_i representing the classification of the image as the number i).

4 Experimental Setup

This section describes how the neural networks were implemented, and informs about the metrics used for the comparison.

4.1 FFNN

The implementation is simplified by the use of the Keras Functional API, which simplifies the process of building the network to selecting the layers and their respective hyperparameters, and then connecting them in the wanted sequence [Chollet, 2020]. The Feed-Forward neural network implemented is a very simple variant of the multi-layer perceptron with a single fully connected hidden layer between the input layer and the output layer. The input layer is a flattened version of the image, meaning we turn a 28×28 image into a single column of $28 \cdot 28 = 784$ nodes. This flattened layer is then densely connected to the hidden layer of 512 nodes. A dense connection between layer A and layer B means that for every node in layer A, we have a connection and a weight to every node in layer B. This structure is very similar to that of a complete bipartite graph. The hidden layer consists of 512 nodes and is kept equal to that of the Bayesian Neural Network described in section 4.2, for purposes of fair comparison. The hidden layer is in turn densely connected to the 10 output nodes. These output nodes will represent the models confidence in the related classification(0 - 9). The structure is visualized in Figure 2.

When choosing hyperparameters for the FFNN model, the Hyperas package was used. This package is a convenience wrapper around the Hyperopt package specially designed for Keras [Pumperla, 2020]. Through this package the hyperparameters were optimized based on the MNIST dataset with a training set of 50000 images and a validation set of 10000. The hyperparameters found in optimization of the largest dropout model, shown in Table 2, were used on all models. This was done to emulate a single model which was implemented in two types, one with and one without dropout, with five training dataset sizes, resulting in 10 models. An exception to this rule was the batch size. The batch size was less important to keep equal in all models, than the number of batches per epoch. This is because with the low learning rate found during optimization, and a high batch size, little learning is made over few iterations. To illustrate, consider the difference between the net with fitting set size of 50000 against one of size 1000. Using the batch size of 128, the larger net would have $\frac{50000}{128} = 391$ weigh adjustments, whilst the smaller net would only run $\frac{1000}{128} = 8$ updates. As learning is applied after every pass, the smaller net would have a inherent large disadvantage which needed to be remedied to focus the results on the project scope. To find the batch size as the closest power of 2 to the batch size which gives a batch count similar to that of the optimized FFNN net of training size of 50000, the following formula was employed: Batch Size(training size) $= 2^{(\lfloor(\frac{1}{2}+\log_2(\frac{\text{training size}}{390}))\rfloor)}$

Hyperparameter	Explored Possibilities	Optimized Selection
Learning rate	$[0.01, 0.0005]$	0.001
Optimizer	SGD, RMSprop, Adam	Adam
Activation Function Hidden layer	relu, elu, sigmoid	sigmoid
Batch Size	16, 32, 64, 128	128
Dropout rate hidden layer	$[0.1, 0.5]$	0.269

Table 2: The optimized hyperparameters by Tree-structured Parzen Estimator on the full training set. Dropout rate and learning rate was selected from a range, rather than a choice.

Hyperparameter	**Explored Options**	**Selection**
Learning rate	0.01, 0.0075, 0.0025, 0.001, 0.0001	0.001
Optimizer	SGD, Adam	Adam
Activation Function Hidden layer	ReLU, Softplus	Softplus
Epoch Count	10, 35, 50, 100, 1000	1000
Batch Size	16, 32, 64, 128, 256, 512	256
Num Samples	5, 10, 20, 25, 50	50
Consensus Selection	Mean, Median	Mean

Table 3: Hyperparameters used when training the Bayesian neural network. Found through manual testing.

4.2 BNN

The BNN had the same architecture as the FFNN described in section 4.1, and was in fact implemented on top of a a with the same structure FFNN. The network's weights and biases were initialized to standard normal distributions.

The BNN implementation made use of two python libraries, Pytorch and Pyro, to perform some modeling and many of the heavier calculations. Pytorch is a comprehensive machine learning framework with powerful data transformation and processing capabilities. The base network that the BNN was built on was built using Pytorch, and Pytorch's built-in mathematical functions were used to setup the network's learning process. In addition, Pytorch was used to load data sets, and to make use of gpu compute to train the network when possible. Pyro is a statistics framework for Python which integrates well with Pytorch. In our work, Pyro was used to create distributions for all the BNN parameters, and its SVI and ELBO classes were used to perform stochastic variational inference when training (called ”guiding”) the models.

As with the FFNN, the hyperparameters used were those that were found to perform best when used with the largest training set. The hyperparameters used were found through manual testing, and are summarized in Table 3. Each model was trained on a flat 1000 epochs without early stopping. Due to the uncertain nature if BNNs, this did not lead to overfitting as is possible with standard FFNNs. A batch size of 256 was used throughout, as it was found to be very beneficial to training efficiency without hurting the model's performance. A large number of samples was used, as it was found to give stability to the model's performance. Adam was chosen as the optimizer as it performed as well or better than the alternatives that were made available through Pytorch. When selecting the consensus of the network, both taking the mean and the median of the sample predictions was considered. Because both performed very similarly, the mean was chosen, as this is what was used in Blundell et al. [2015].

Since the input images were normalized to be in the range [0, 1], initial distributions were selected to be normal distributions with expected value and standard deviation random numbers from the same range.

4.3 Classification reports and accuracies

For each of the networks trained on 50'000 instances, a classification report containing the accuracy of the network and the precision, recall, F1-score, and support of each class was created with the `sklearn.metrics.classification_report` method from the sklearn library [Pedregosa et al., 2011]. Precision and recall was measured in order to compare the networks on their ability to only identify correct instances for each class and their ability to find all the correct instances per class, respectively [Sokolova and Lapalme, 2009]. For classifiers, having both a high precision and recall is important as only obtaining one does not mean the classifier is any good. A classifier can obtain a high precision for a class by only classifying one instance as the given class and classifying this instance correctly. Furthermore, a high recall can be achieved for a class by evaluating every instance as that class. Thus, the F1-score is measured as this is a weighted harmonic mean of precision and recall, normalized to a value in the interval [0,1], where 1 is the best possible score. Furthermore, the accuracy score is mathematically equivalent to the micro average scores, i.e., the score calculated from the aggregate contributions of all classes, for precision, recall, and F1-score of the network [Pedregosa et al., 2011], and indicates how the network performs overall.

4.4 Entropy

Entropy was first formalized by Rudolf Clausius in 1865 in relation to thermodynamics, and explained the gradual decline in intensity of heat, pressure and density over time [Cropper, 2004]. In 1948, Shannon extended the concept to be a measure of uncertainty in probabilistic distributions [Shannon, 1948]. The application and formalization of entropy in information systems, also known as Shannon Entropy, is widely used as the primary measurement of uncertainty, and therefore also certainty, in the classifications of a neural network. In the project, entropy is the measure used to view the uncertainty showed by the models predictions. For each presented image, the model creates a probability distribution reflecting the models probability of the classifications (0-9). A distribution with high uncertainty, e.g. a prediction which cannot distinguish between any and therefore setting a 0.1 value on each class, will have the maximum entropy of $\ln 10 \approx 2.3$. In contrast, a model prediction which is completely certain on a single class, will have no entropy. From this, we can view the models' ability to express uncertainty through the lens of their predictions' entropy. As large entropy of a prediction would then signify large uncertainty, entropy is useful in that it may be used to determine outliers and data which should be passed to further analysis or post-processing.

5 Results

This section will first present the results from the classification reports, then it will present different results given by the entropy of the models in different scenarios.

5.1 Classification reports and accuracies

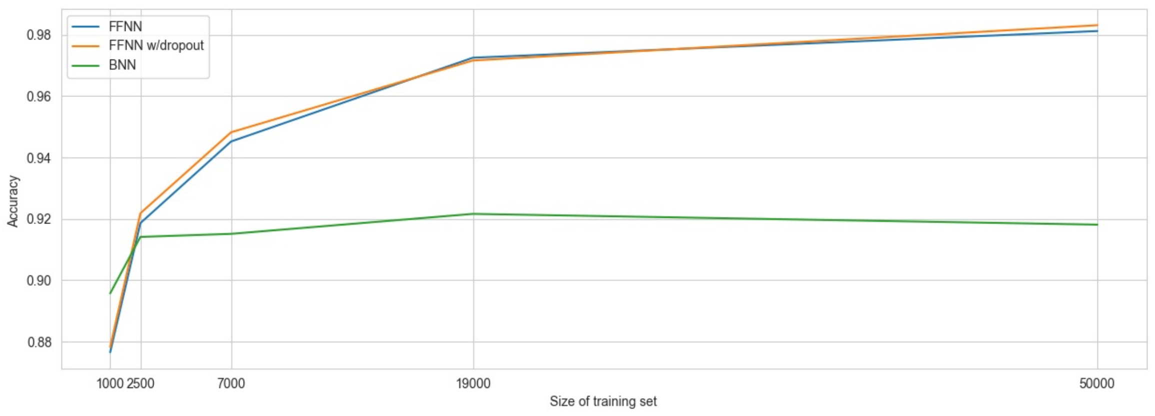


Figure 3: How the accuracy on the MNIST test set change with different training set sizes for the FFNN, the FFNN w/dropout, and the BNN.

The results from the classification reports are shown in Table 4. This table shows that both of the FFNNs perform

	FFNN			**FFNN w/dropout**			**BNN**			**Support**
	Precision	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**	
0	0.98	0.99	0.99	0.98	0.99	0.99	0.95	0.98	0.96	980
1	0.99	0.99	0.99	0.99	0.99	0.99	0.97	0.98	0.98	1135
2	0.98	0.98	0.98	0.98	0.99	0.98	0.91	0.89	0.90	1032
3	0.97	0.98	0.98	0.97	0.99	0.98	0.88	0.90	0.89	1010
4	0.98	0.98	0.98	0.98	0.98	0.98	0.92	0.89	0.90	982
5	0.98	0.97	0.98	0.98	0.98	0.98	0.91	0.87	0.89	892
6	0.98	0.99	0.98	0.98	0.98	0.98	0.90	0.95	0.93	958
7	0.98	0.98	0.98	0.98	0.98	0.98	0.96	0.91	0.93	1028
8	0.98	0.97	0.98	0.99	0.97	0.98	0.91	0.88	0.89	974
9	0.98	0.98	0.98	0.98	0.97	0.98	0.87	0.92	0.89	1009
Accuracy	0.98			0.98			0.92			10 000

Table 4: Metrics obtained on the MNIST test set for the FFNN, the FFNN w/dropout, and the BNN trained on 50 000 examples with a validation set of 10 000. Furthermore, the support for each class was the same for each network and is given in its own column.

well for all the classes with a F1-score of 0.98 or 0.99 for each class, and thus they achieve a high accuracy score of 98%. The BNN performs a bit poorer overall, with an accuracy of 92%. For every class, both of the FFNNs have higher precision and recall than the BNN, while the precision and recall of every class is approximately the same for both of the FFNNs. Utilizing the BNN in stead of one of the FFNNs will give approximately a 8-9% reduction of the F1-score for the classes 2, 3, 4, 5, 8, and 9, approximately a 5% reduction for the classes 6 and 7, and approximately a 1-3% reduction for the classes 0 and 1.

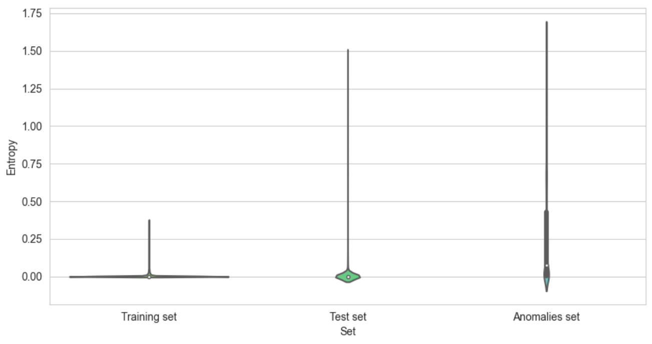
Figure 3 shows how the accuracy of the FFNN, the FFNN with dropout, and the BNN change with varying training set sizes when run on the MNIST test set. The figure shows that both of the FFNNs have a lower accuracy than the BNN when a training set size of 1'000 instances is utilized, and that they have a higher accuracy for the explored training set sizes of 2'500 instances or more. Furthermore, the accuracies of the FFNNs grow more rapidly than that of the BNN, resulting in a growing difference between the accuracies of the FFNNs and the BNN. In addition, the accuracy of the BNN drops from right above 92% to right below 92% when increasing the training set size from 19'000 instances to 50'000 instances, further increasing the difference between the accuracies. The figure also shows that the accuracies of the FFNNs are within 0.5% of each other.

5.2 Entropy

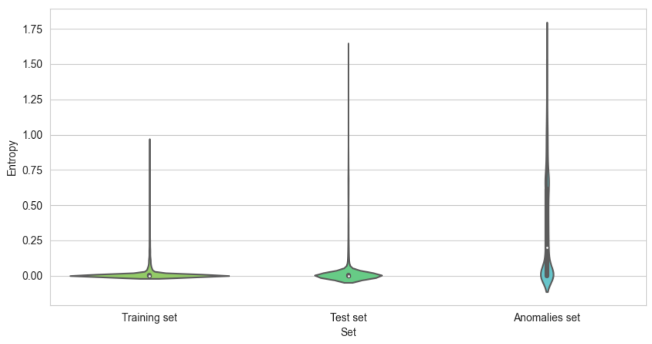
Figure 4 shows the entropy of the largest regular FFNN model, the largest FFNN with dropout, and the largest BNN model. Figure 4(a) shows that the predictions of the FFNN on the training set have very low entropy, with 98.1% predictions accuracy with a entropy lower than 0.05, and 93.8% for the test set. On the anomaly set, the model struggles to notice the different inputs and react accordingly with uncertainty. The entropy is larger than the test set and the training set, but the entropy of the predictions are still very low, with a median of 0.08. The FFNN with dropout, Figure 4(b), shows the same tendencies as the FFNN, however it has higher entropy in general. The median entropy of the test set was 0.002, showing a high confidence in the predictions of the test set. For the anomaly set the median entropy was 0.25, which is is a significantly higher change than that of the regular FFNN.

Compared to the other networks, the BNN model, Figure 4(c), shows a clearly higher entropy in the anomaly set, with a median of 1.9, which is an achievement seeing as $\ln 10 \approx 2.3$ is the optimal score to show complete uncertainty.

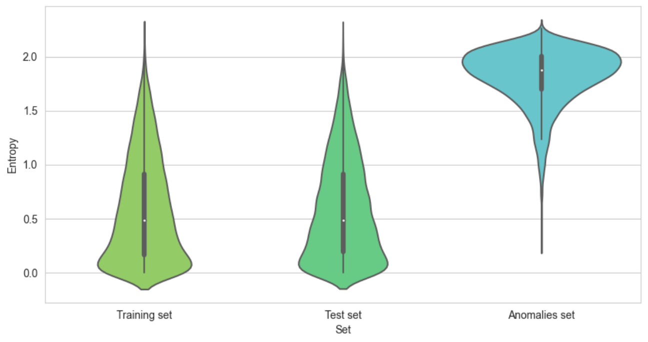
In Figure 5, we can see that the model confidence does not significantly correlate with size of training set. This is reasonable as a training set of 1000 over many epochs is sufficient to fit the model, and utilizing the early stopping, overfitting is mitigated. Hence, the size of the training data does not materially impact the models confidence in its predictions.



(a) FFNN model

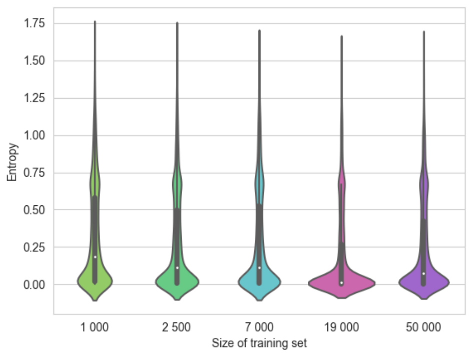


(b) FFNN model with dropout

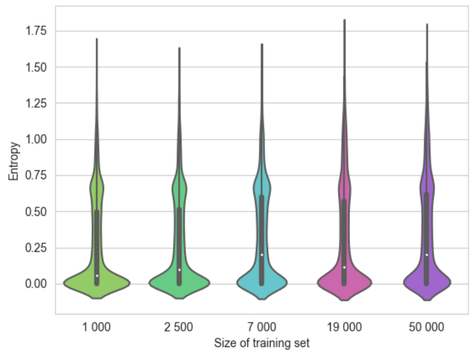


(c) BNN model

Figure 4: Entropy of predictions on the MNIST training and test set and the notMNIST anomaly set given by (a) the FFNN, (b) the FFNN with dropout, and (c) the BNN.



(a) FFNN models



(b) FFNN models with dropout

Figure 5: Entropy of predictions on the notMNIST anomaly set, with varying training set sizes, on FFNN models and FFNN models with dropout.

6 Discussion

Based on the classification reports presented in Table 4 and discussed in Section 5.1, both of the FFNNs perform better than the BNN when evaluating on precision, recall, F1-score, and accuracy, when the networks are trained on 50000 instances and utilizes a validation set of 10000 instances. Figure 3 indicates that both of the FFNNs performs better than the BNN when a training set of 2500 instances or more is utilized. BNNs, however, generally needs less data than many neural networks, which may be the reason that the BNN performs better than the FFNNs on the smallest explored training set. Furthermore, this property might explain why the accuracy of the BNN does not increase significantly when increasing the training set size from 2500 instances. Thus, based on the results from the explored training set sizes, employing the BNN can be beneficial for small training set sizes in regards to accuracy, however both of the FFNNs perform significantly better for larger training sets.

As discussed earlier, the BNN is better at showing uncertainty than the FFNNs. Thus, the BNN will also be more uncertain, which may be one of the reasons the FFNNs have a higher accuracy than the BNN for larger training sets. The FFNNs grow more and more confident in their decisions as the size of the training set increases, while the BNN has approximately the same certainty for all the training sets. This uncertainty can affect the BNN to misclassify more instances than the FFNNs. By misclassifying instances, both the recall for the target class, as well as the precision for the selected class, are reduced. The reduction in precision and recall negatively affects both the F1-score of the affected class and the total accuracy of the network. Thus, the uncertainty of the BNN can be a reason for its poorer performance on precision and recall, and thus F1-score, on every class of the test set, and its poorer accuracy on the test set.

Another possible reason that the BNN did not perform as well as the FFNNs, could be due to a non-optimal implementation. Blundell et al. [2015] report that the accuracy of their method Bayes by Backprop can rival a dropout with similar accuracy to ours. Thus, it is probable that the BNN implemented in this project was sub-optimal, and another implementation might achieve better results.

In Figure 4(a), the slightly higher entropy shown in the anomaly predictions by the regular FFNN may stem from the different style of the datasets, where notMNIST are typed and MNIST are hand written. The increase in entropy is very low and shows from this a struggle in letting model confidence impact the predictions, which is in line with the dropout model reasoning presented by Gal and Ghahramani [2016]. The model does however, accurately classify the test set at more than 98%, which would indicate that the model confidence in the test set is justified. The FFNN with dropout, Figure 4(b), shows the same tendencies as the FFNN, however it has a much higher entropy for the anomaly set, indicating a more reasonable confidence in the face of anomalies. Even though it shows a higher entropy for the anomaly set, and higher entropy in general, the model accurately classifies the test set at a similar rate as the regular FFNN. These results indicate that the implementation of the probabilistic element of dropout yields a model which is comparable in its classification ability, whilst being better suited for noticing anomaly introduction.

The Figure 4(c) is very interesting for the intended future work of the project. The BNN model shows a clearly higher entropy in the anomaly set, with a median of 1.9, which is an achievement seeing as $\ln 10 \approx 2.3$ is the optimal score to show complete uncertainty. In addition to this, the clear split between the anomalies and the MNIST data sets shows its validity as a foundation for anomaly detection. Another interesting point of Figure 4, is the very similar shape of the entropy distribution among the training and test sets. This indicates that the model does not suffer from overfitting and has learned a suitable general function for classifying MNIST images.

7 Conclusion

This paper addresses a simple FFNN and the introduction of probabilistic methods to it. The methods introduced are dropout, which is implemented following Gal and Ghahramani [2016] in the construction of the model to create a Bayesian approximator, and a BNN, implemented using Bayes by backprop [Blundell et al., 2015]. Each of these networks were trained on different sizes of the training set and were compared with each other on their accuracy on the MNIST test set, as well as their ability to show uncertainty; evaluated based on the entropy of the output probabilities. Furthermore, the networks that were trained on a set of 50000 instances were evaluated on their precision, recall, F1-score, accuracy and entropy.

These evaluations showed that both of the FFNNs had better accuracy than the BNN for all networks trained on 2500 instances or more. Furthermore, the evaluations showed that the precision, recall, and F1-score of the FFNNs trained on 50000 instances was better than the BNN for all the classes in the test set. However, BNN had a higher entropy than the FFNNs on all the explored sets, and had a significant increase in entropy from the MNIST train and test set to the set of anomalies of the notMNIST data set. The higher entropy indicates that the BNN is more capable at showing uncertainty than the FFNNs, and could thus be better suited for classification tasks were this ability is essential, something which can be seen in Figure 4(c). While the dropout model also showed increased entropy on the anomaly set, indicating a better ability to avoid over-confidence, its usage in anomaly detection is limited without a somewhat linear separation of the anomaly set from the test and training sets, such as in Figure 4(c) where one can set a boundary on an entropy of 1.5 and have a good separation of anomalies from valid images, with little loss of accuracy. However, in Figure 4(b), we can see no such separation. This would indicate a less useful model for anomaly detection, but an interesting subject for further research would be to expand the model to more hidden layers and a higher dropout rate and view if the model shows a clearer separation. A separate path would be to rather utilize state-of-the-art model classes such as CNN, and research the possibility to implement probabilistic characteristics to a CNN. Continuing on this, one could attempt to create a fully-fledged anomaly detection model by using the outlined possibilities.

Bibliography

J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. *TProc. of the 30th International Conference on Machine Learning*, 2013.

C. Blundell, J. Cornebise, K. Kavukcuoglu, and Daan Wierstra. Weight Uncertainty in Neural Networks. In *Proceedings of The 32nd International Conference on Machine Learning*, volume 37 of *ICML'15*, pages 1613–1622. PMLR, 2015.

Y. Bulatov. notMNIST dataset, September 2011. URL http://yaroslavvb.blogspot.com/2011/09/notmnist-dataset.html.

Y. Bulatov. Index of /upload/notMNIST, n.d. URL http://yaroslavvb.com/upload/notMNIST/.

François Chollet. Keras documentation: The functional api, Dec 2020. URL https://keras.io/guides/functional_api/.

William H. Cropper. *The Road to Entropy Rudolf Clausius*, page 93–105. Oxford University Press, 2004.

K.-L. Du and M. N. S. Swamy. *Neural Networks in a Softcomputing Framework*. Springer, 2006.

Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *ICML'16*, pages 1050–1059. PMLR, 2016.

M.I. Jordan, Z. Ghahramani, T.S. Jaakkola, and et al. An introduction to variational methods for graphical models. *Machine Learning*, 37:183–233, 1999. doi: 10.1023/A:1007665907178. URL https://doi.org/10.1023/A:1007665907178.

Y. LeCun, L. D. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Muller, E. Sackinger, P. Simard, et al. Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective*, pages 261–276, 1995.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Y. LeCun, C. Cortes, and C. J. C. Burges. The MNIST Database of Handwritten Digits, n.d. URL http://yann.lecun.com/exdb/mnist/.

David J C Mackay. Probable networks and plausible predictions — a review of practical bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6(3):469–505, 1995. doi: 10.1088/0954-898X_6_3_011. URL https://doi.org/10.1088/0954-898X_6_3_011.

T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. *CoRR*, abs/1412.1897, 2014. URL http://arxiv.org/abs/1412.1897.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

M. Pumperla. Hyperas. https://github.com/maxpumperla/hyperas, 2020.

S. R. Saucedo. Bayesian Neural Networks for Classification. Master's thesis, Florida State University, 2007.

Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing and Management*, 45(4):427–437, 2009.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

G. P. Zhang. Neural Networks for Classification: A Survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 30(4):451–462, 2000.