UPPSALA
UNIVERSITET

UPTEC STS 19028
Examensarbete 30 hp
Juni 2019

Finding mislabeled data in datasets

A study on finding mislabeled data in datasets by studying loss function

Salam Jadari

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Finding mislabeled data in datasets

Salam Jadari

The amount of data keeps growing thus making the handling of all data to an extensive task. Most data needs preprocessing in different ways and to ease this process methods and techniques are required.
The aim of the thesis is to find mislabeled data in datasets by studying training related metrics for individual data-points. The metric studied in this thesis is the loss function.
Experiments were made on MNIST and CIFAR10 datasets. First a CNN was trained as one part of the filtering process. The individual losses were then obtained and stored. The second part consisted of distance measuring these losses. Both the Euclidean and Manhattan distances were calculated for each data-point to the median class loss. The hypothesis of the thesis is that a greater distance to the median class loss is associated with more uncertainty of the given label. Datsets that were studied was the MNIST and CIFAR10.
The results shows that it is possible to find mislabeled data by studying individual loss functions.

Handledare: Johan Nyberg
Ämnesgranskare: Michael Ashcroft
Examinator: Elísabet Andrésdóttir
ISSN: 1650-8319, UPTEC STS 19028

Populärvetenskaplig sammanfattning

Mängden data i världen växer ständigt i en ökande takt. Mobiltelefoner, molntjänster och framförallt internet har möjliggjort detta. All denna data lagras bland annat för att skapa mervärde åt konsumenter. Stora företag men också vetenskapliga institutioner drar nytta av all data. Mer data innebär bättre underlag för modeller. Detta tillsammans med den beräkningskapacitet dagens datorer klarar av gör att möjligheter tillåts som tidigare inte var aktuella. I takt med att data växer krävs också verktyg och metoder som underlättar hanteringen av all data, vilket denna uppsats önskar bidra till.

Olika data behöver ibland bearbetas bland annat genom märkning. Detta sker i vissa fall av människor eller utvecklade verktyg som har till uppgift att underlätta denna process. Det händer då att märkning blir fel och att modeller som man använder datan till att träna upp påverkas till det sämre. Samtidigt kan kvaliteten på datan vara en fråga om kostnad då bland annat manuell märkning ofta är en dyr process. Det blir då en avvägning att besluta hur mycket fel som får finnas i ett dataset.

Denna uppsats introducerar metoder vilka man kan applicera för att identifiera felmärkt data. Detta görs genom att undersöka individuella datapunkters träningsindikatorer och jämföra med medianindikatorer för den specifika datapunktens märkning. Indikatorn som används i denna studie är utfallet av förlustfunktionen för individuella datapunkter. Förlustfunktionen visar hur en modells utfall relaterar till märkningen av datapunkten. En låg förlustfunktion är önskvärd för individuella datapunkter då detta indikerar att modellen generaliserar väl. För datapunkter med hög förlustfunktion kan man misstänka att märkningen är felaktig eller att modellen inte lyckats generalisera tillräckligt väl.

Förlustfunktionen lagras under 100 epoker för alla datapunkter. Metoderna bygger på att standardisera och normalisera förlustfunktionen för datapunkterna. Därefter beräknas det Euklidiska och Manhattanavståndet till klassens median. Uppsatsen bygger på antagandet att ett stort avstånd innebär en felmärkning för datapunkten. Dataseten som studeras i denna uppsats är MNIST och CIFAR10. Det första är ett dataset med handskrivna siffror och det andra är tio olika klasser såsom bland annat bilar, lastbilar, hundar och katter. Måttet för hur bra en metod presterar är valt till arean under mottagarens operativa egenskaper, även förkortat ROC AUC.

Uppsatsen visar att det är möjligt att identifiera felmärkt data genom att undersöka träningsindikatorer. Distansen visar sig vara ett potentiellt mått på att upptäcka felmärkt data i samtliga experiment. Den metod som visar bäst resultat är standardisering av förlustfunktion med Euklidiskt avståndsmått.

Table of content

1. Introduction

There are several reasons why data has grown in recent years. Internet, mobile devices and cloud computing are some of them, just to mention a few. Big companies store all kinds of data. The collected data in the world is consequently growing at an exponential rate. Dumbill [1] summarizes the definition of Big Data from public discourse as the notion that we might be able to compute our way to better decisions. Big companies like Facebook, Amazon, and Google make use of machine learning models in their products. In order to give customers better user experience, companies use the power of machine learning models. Gartner [2] stated that a majority of big companies either have invested or are planning to invest in big data. It's also a consequence of the possibility to be able to store data as the technology in storing data is getting better [3]. Machine learning has also increased in medical diagnostics and scientific computing among other fields. All these models depend on data in the training phase to be sufficiently good and is also a reason why creators of machine learning models store data.

There are many indications that data will play a big role in the success of companies in the future. As the amount of data is increasing, handling all the information will get increasingly more complex [4]. One example is in categorizing and labeling data. Some kinds of data need processing in one way or another and others do not. Companies, scientists and other people who handle all the data are in need of tools and methods that help with processing all information as well as other tools. The better the data, the better the result can be expected from the machine learning models in terms of model accuracy and other metrics. To get the most out of models, one needs to make sure the training phase data is correctly labeled and categorized. When datasets contain many errors, the need for more training data is required in contrast to more clean datasets [4] [5].

Cleaning and processing data are often an expensive task in terms of cost and man-hours. Almost always, when cleaning and processing are done on a large amount of data, errors slip through. Many kinds of errors can occur, including human errors. With the aim to screen datasets from erroneous labels, one can study metrics in the training process for each individual image. This study aims to see if it is possible to find erroneous labels by considering individual losses, see section 2.1.2, over time as time-series and use distance measuring techniques to find erroneously labeled data.

1.1 Problem description

This section begins with a short taxonomy part and then presents the problem description. Erroneously labeled data is common in big datasets and there are several reasons for why it occurs. Since labeling data is an expensive process in terms of man-hours and cost, creators of datasets have to make decisions of what is an acceptable level of errors, which explains the commonness of errors [5]. Also, the type of errors differs from one dataset to another. The pixel values of an image are considered feature variables while the label of the image is the target variable.

All the errors affect the accuracy of machine learning models at different levels. There are several definitions of what error and noise are in the context of datasets. One definition draws a line between two main categories; feature and class noise [4] [5]. Feature noise considers the features of the data while class noise considers the label of the data. Feature noise could be a small Gaussian noise in the feature variables. Label noise is when a specific target variable of an image is wrong. Either a label that is switched with another label in the dataset or when the image belongs to a class outside the classes of the dataset. One example of label noise would be if there is a bike in a dataset which only should contain cats and dogs.

Brodley and Friedl [6] categorize errors in three classes. Subjectivity errors, data-entry errors and inadequacy errors. Subjectivity errors occur when instances are ranked like in disease severity. In cases when many experts most likely won't have the same opinion, subjectivity errors can occur. Data-entry errors are often associated with human factors. When humans are given the task to classify a big amount of data, data-entry errors almost always will occur. Data-entry error is a common source of class label errors. The last type of error that the authors mention, inadequacy errors, is instances when there is not enough information to classify. The latter type is common in medical diagnosis. Brodley and Friedl also mention that by removing the class label errors, one can increase the accuracy of the model. By removing feature variable errors, the accuracy on the target data with the given labels will decreases if the same feature variable errors occur in the test data.

1.2 Purpose

The loss function in CNN's is calculated from the activation layers during training. When an instance is wrongly labeled, the corresponding loss is high. CNN’s will be described in parts later in this thesis. The last layer in CNN is called the activation layer. If a CNN trained on classifying handwritten digits, the number of neurons in the activation layer would be 10 (0 to 9). When a CNN classifier is making predictions with high uncertainty, the activation would be evenly distributed in comparison if the classifier is predicting with low uncertainty. There will be an in-depth explanation in the theory chapters on the activation layers and loss function.

The hypothesis of this study is that it is possible to find erroneously labeled data by studying individual losses over time and then measuring the distance to median class losses. This thesis will present four methods. Therefore, the aim of the study is to compare methods and see how well the methods detect wrongly labeled data in datasets. The type of error is what Verleysen et al [5] describes as label noise. Moreover, the label noise can be divided into two subcategories. One in which the label is switched with another label in the same dataset and the other type is when an image has a label outside the dataset. This thesis will study the former type of label noise.

My research questions are as follows:

- Is it possible to find erroneous labeled data through studying loss function? (RQ1)
- Which of the methods provides the best predicting power in terms of AUC in finding erroneous labels for each error level and dataset? (RQ2)

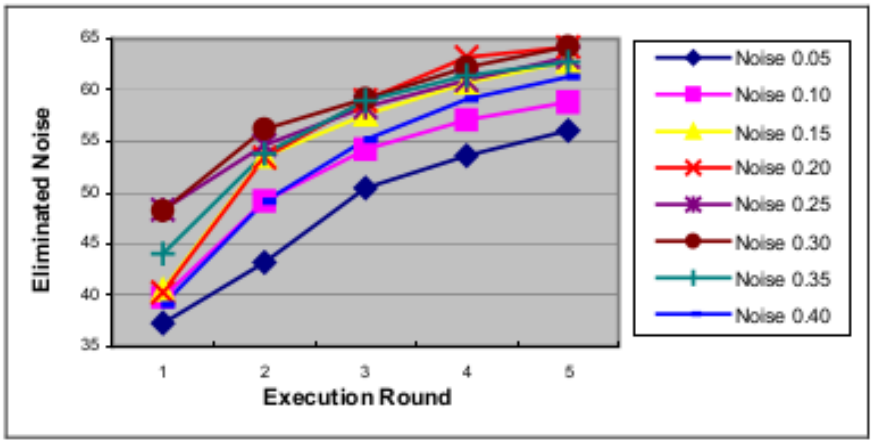
Due to the scope of the study, there are some delimitations. In chapter future work there will be described how one might proceed in studying erroneous labels with similar methods. The limitations are:

- Only two datasets and two error levels are studied
- The number of epochs is fixed
- The type of error is limited to errors in the label when labels have been switched which is one type of label noise

1.3 Related work

Brodley and Friedl [6] did an extensive study on finding mislabeled examples in datasets. They did so by first splitting the dataset into N sets. Then they trained decision tree and K-NN classifiers on each set. Each classifier voted on the other examples that did not belong to the same set. The authors trained a model without injecting errors and obtained a baseline accuracy for different datasets. After injecting different levels of errors, they concluded that when injecting up to 20% errors, and applying their method they achieved baseline accuracy which is what was obtained before the errors were injected. They also concluded that when the error level was high enough the classifiers were not able to accurately classify on some of the studied datasets.

Zhu et al [7] also partitioned the datasets. They extracted a good set of rules from all partitions and classified the whole dataset based on the extracted rules. Each image got an error count value based on if the classifiers predicted the images as errors or not. The idea is to remove noise in rounds instead of removing them all at once. They evaluated their method on two datasets. Lastly, they implemented a threshold scheme to identify the errors. One of the threshold schemes requires a value which is an estimate of how much errors there is in the dataset, called adaptive threshold scheme. They concluded that their approach was robust and effective in identifying errors and improving classification accuracy. Figure 1 shows how they were able to remove noise over rounds on the Adult dataset, which is a dataset with attributes on people. The classification task is to predict whether a person's income is more than $50k annually. The x-axis shows the rounds of cleaning and the y-axis shows the percentage of found errors until that specific round. The authors evaluated 8 error levels.



Figur 1. Percentage of found errors as a function of execution round.

Zhang [8] proposed an improved method which builds on Brodley and Friedl's idea. Instead of using decision trees and K-NN, Zhang used a convolutional neural net as a classifier. The classifier acted as a filter before training the machine learning model in question. Figure 2 shows how the procedure of the classification and training.

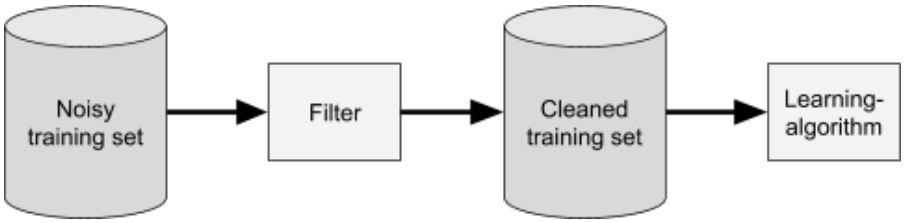


Figure 2: Zhang's training and classification procedure.

Zhang evaluated the improved method on MNIST and CIFAR10. The method found 675 errors in MNIST dataset and 118 errors in CIFAR10 dataset. Table 1 shows what digits were found to be mislabeled in the MNIST dataset.

Table 1: Zhang's found errors in MNIST dataset by digit

Digit	0	1	2	3	4	5	6	7	8	9
Total number of images	7000	7000	7000	7000	7000	7000	7000	7000	7000	7000
Found mislabeled images	44	62	118	17	147	74	70	51	61	31

Frénay and Verleysen [5] mention that when dealing with outliers or anomaly detection, one can often use the same techniques to find mislabeled data. They mean that outliers may be errors or anomalies, which makes the same problem domain suitable for finding erroneous labels. However, mislabeled data are not always outliers or anomalies which could be a subjective concept [5].

2. Theory

In this chapter, we first present a background on general statistical modeling. Secondly, neural networks are presented. Lastly, time-series and score metrics are described.

2.1 Background on statistical modeling

In this section, theory will be presented on statistical modeling in general terms.

2.1.1 Parametric models

The goal of a statistical model is to map an input variable X to an output variable Y. Parametric models (1) make assumptions about the functional form of the output variable with parameters, w.

$$\hat{y} = f(x, w) \tag{1}$$

Parametric models have a fixed number of parameters independently of the size of data. One parametric model is the binary logistic regression model (2) which is a common simple classifying model.

$$\hat{y} = f(x, w) = \frac{1}{1 + e^{-w^T x}} \tag{2}$$

Where w are the parameters of the model and x is the input variables, except x_0 which is a dummy variable that always has the value 1 so as to have a constant term. One example of a logistic regression model is whether or not a person is obese or not which is a binary classification problem. One can assign a data-point to 0 if $\hat{y}$ is below an introduced cut-off value, P_{co}. If the value of $\hat{y}$ is greater than P_{co}, the outcome is 1 (3).

$$P(obese) = \begin{cases} 1, & if\ f(x, w) > P_{CO} \\ 0, & if\ f(x, w) \leq P_{CO} \end{cases} \tag{3}$$

2.1.2 Loss function

A loss function (4) measures the penalty of a model's outcome given the feature and target variables.

$$L(x, y, w) \tag{4}$$

The goal of the loss function is to get information about how to change model parameters to be able to make more correct classifications. The loss function for logistic regression is described in (5).

$$L(x, y, w) = \begin{cases} -\log\big(f(x, w)\big), & if\ y = 1 \\ -\log\big(1 - f(x, w)\big), & if\ y = 0 \end{cases} \quad (5)$$

If a classifier is given output of 0.9 and the target variable is 1, the corresponding loss will be close to 0.1. If instead, the target variable is 0 the loss will be 2.3. On the other hand, If the target variable is 0, the loss value for $\hat{y} = 0.9$ would be 2.3 and $\hat{y} = 0.1$ would be 0.1. The point is that better classification gives lower loss value and vice versa.

2.1.3 Training as an optimization

In order for the model to improve over the training data, an optimization function is required. The goal of the training is to improve model parameters based on the objective function (6).

$$\widehat{w} = argmin_w\, f(w) \quad (6)$$

f(w) is the loss-function in many machine learning models. The weights are then updated e.g. with a first-order optimization method (7)

$$\widehat{w}_i = w_{i-1} - \eta \nabla L(x, y, w) \quad (7)$$

Where η is step-size.

2.1.4 Bias-Variance

The loss is calculated on training data, but we want a model that generalizes to new data. Therefore, we want to minimize the expected error on new data. There are three kinds of error that occur when training machine learning models. These are Bias error, variance error, and irreducible error. Bias error is associated with underfitting. When bias error is high, the model won't be able to model the relationship between features and target output. On the other hand, variance error is associated with overfitting. A high variance error means it knows the relationship between the feature variables and target variables of the specific dataset. A high variance model has difficulties in generalizing new data points correctly. Irreducible error is related to the randomness inherent in the system being modeled. No matter how good the model is, there will be some irreducible error. Bias and variance are both related to the complexity of a model. For a given amount of data, as model complexity increases bias will decrease and variance increase (and vice versa when model complexity decreases). The trade-off between these kinds of errors is of great importance when creating classifiers and other kinds of models. The total error, also known as the generalization error, is what should be minimized. Figure 2 shows the relationship between the different errors and the total error. In optimal situations, one should train the model so that the total error is close to its minimum.

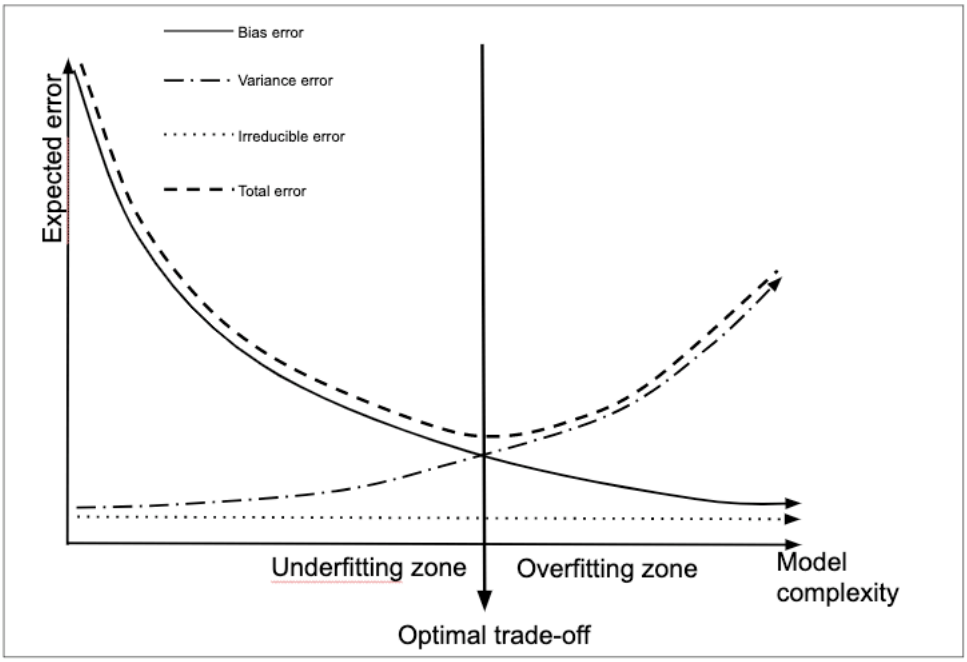


Figure 2: Expected errors in training phase as a function of model complexity.

2.1.5 Regularization

Regularization techniques permit us to control the complexity of a model via the adjustment of regularization parameters, and without adjusting the model's parametric form. The specific regularization techniques used will be later be described in section 2.2.3.

2.1.6 Validation

The model adjusts the parameters/ weights only to the training set. A validation set is used to measure the accuracy and the generalizability of the model. The accuracy and loss of the validation set should follow the same trend as the accuracy and loss of the training set. If the accuracy decreases for the validation set, it could mean that the model is overfitting and not being able to generalize to new data-points. Figure 3 illustrates a possible case of overfitting, where improvements on performance in the training data is unlikely to generalize to validation data. Plot A represents the preferable decision boundary and Plot B represents an overfitted model.

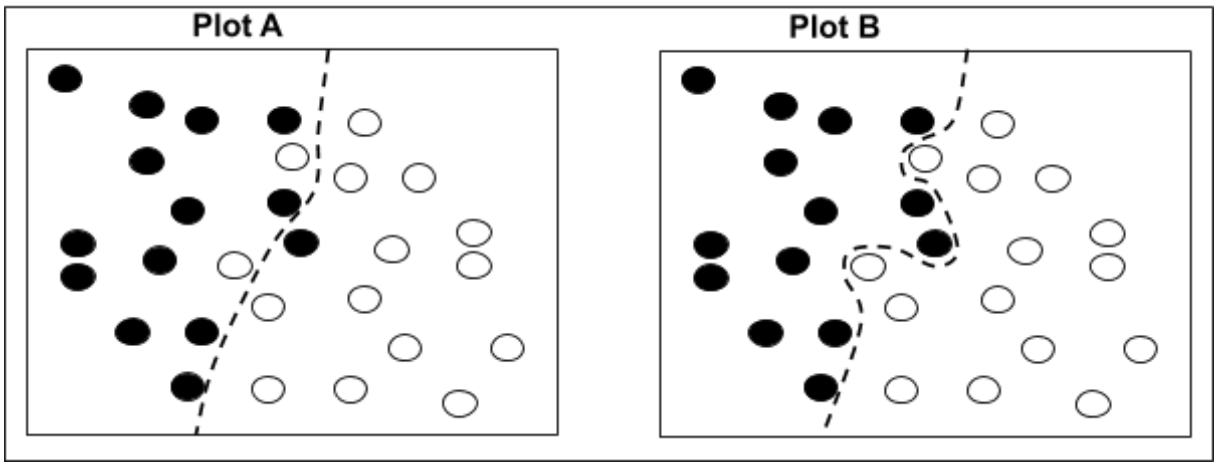


Figure 3: Decision boundary when validation loss increases while validation accuracy is stationary.

2.2 Feed-forward neural network (FFNN)

A simple FFNN consists of an input layer, hidden layers, and an output layer. The feed-forward refers to the flow of data which only flows in one direction, forward. The layers consist of neurons. Each neuron has a weight associated with it. Each layer also has a bias unit. Figure 4 describes a simple FFNN with an input layer, one hidden layer, and an output layer.

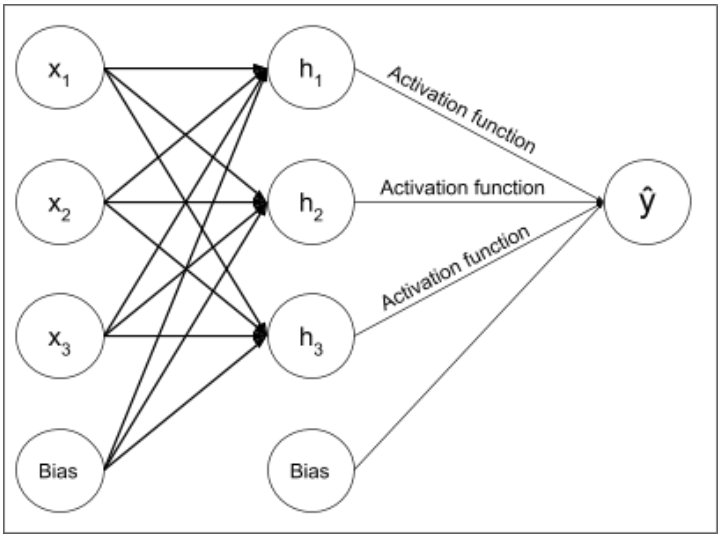


Figure 4: FFNN with one input layer, one hidden layer and one output layer.

The neurons in the input layer and the hidden layer are denoted x_i and h_i respectively. The output $\hat{y}$ is calculated as a function of the hidden layer neurons described in (8) (Persson 2018).

$$\hat{y} = f(\sum_{i=1}^{n} w_i x_i + bias) \tag{8}$$

The function f is called the activation function. All lines drawn in figure 4, except the ones that is connected to the output, represent weights. The weights are denoted wi. The inputs are denoted xi. Activation functions will be presented in more detail later.

2.2.1 Dense layers

Dense layers, also called fully connected layers, are the layers at the end of the network. Each dense layer neuron is connected to all neurons in the previous and next layer [9]. The last dense layer is the classification layer.

2.2.2 Training the FFNN

The main components of the training will be described in further depth below starting with the Optimization function and learning rate.

Optimizer and learning rate

The training process starts with computing the activations in the last layer by inserting the input/ image into the network. Each epoch is finished when all training instances have been inserted into the network. By comparing the ground truth for that specific training instance with the activation outcome, a loss is calculated. The loss value represents an average of a number of training instances since it is more computationally efficient and more accurate in terms of finding a minimum. The training instances which together contribute to a loss is called the batch. The loss is often large at the beginning of the training. The goal is to minimize the loss with respect to the model weights. It is described in (9).

$$L(w) = \sum_{i=1}^{l} L(w, D) \rightarrow \min w \tag{9}$$

The last part of a training instance is updating weights by minimizing the loss depending on the optimization method. If gradient descent is chosen as the optimization function, the weights are updated according to (10).

$$w^t = w^{t-1} - \eta \nabla L(w^{t-1}) \tag{10}$$

η is the learning rate. Computing all gradients are computationally expensive, and in some cases infeasible. The stochastic gradient descent fixes the problem by computing the gradient for each picture. The advantage is that we can update the weights after each image or data point. The disadvantage is that it can be noisy and slow to converge. The stochastic gradient descent is described in (11).

$$w^t = w^{t-1} - \eta \nabla L(w^{t-1}; x_i; y_i) \tag{11}$$

In stochastic gradient descent, $i_1,...,i_m$ is different parts of the dataset between 1 and l. The mini-batch stochastic gradient descent updates weights based on a subset/ batch of the dataset instead of individual images as in stochastic gradient descent, described in (12).

$$w^t = w^{t-1} - \eta \frac{1}{m} \sum_{j=1}^{m} \nabla L(w^{t-1}; x_{i_j}; y_{i_j}) \tag{12}$$

This optimization method is less noisy than stochastic gradient descent since it calculates the updates to the weights on batches of the dataset instead of each individual image, as in gradient descent. Momentum-based methods introduce a vector h which we also update in each iteration. Suppose g is the gradient approximation in each step. Then h is updated according to (13).

$$h_t = \alpha h_{t-1} + \eta_t g_t \tag{13}$$

Where α is a coefficient, often set to 0.9. The weights are then updated in accordance with (14).

$$w^t = w^{t-1} - h_t \tag{14}$$

The momentum vector makes sure that repeated directions term of the gradient is increased in value and directions which are rarely calculated is decreased. The fluctuations in the wrong directions are then decreased and the movement in the right directions is increased. Momentum based optimizers are often sensitive to the learning rate. RMSprop is an optimizer with an adaptive learning rate for each weight. Equation (15) describes what is multiplied with g instead of only η as in e.g. momentum-based methods.

$$\Delta w_t(t) = -\frac{\eta}{\sqrt{G_i^t(t) + \varepsilon}} \tag{15}$$

Where G(t) is described in (16).

$$G_t(t) = \beta G_i^{t-1} + (1 - \beta) g_{ti}^2 \tag{16}$$

In RMSprop the learning rate adapts and makes the optimizer more insensitive to the chosen learning rate. Adam (Adaptive Moment Estimation) consists of both an adaptive learning rate and a momentum [10]. The adaptive learning rate makes the optimizer robust to different learning rates. It consists of three hyper-parameters; which are the learning rate η and the exponential decay rates β_1, and β_2. The optimization function includes

calculations of the first and second-order moment of the gradient. The function is described in (16) to (20).

$$m_t = \beta_1 \times m_{t-1} + (1 - \beta_1) g_t \tag{16}$$

$$v_t = \beta_2 \times v_{t-1} + (1 - \beta_2) g_t^2 \tag{17}$$

$$\widehat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{18}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{19}$$

$$w_t = w_{t-1} - \eta \frac{\widehat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}} \tag{20}$$

In (16) and (17) the mean and the uncentered variance is calculated. (18) and (19) shows the calculation of the bias-corrected estimators of the mean and variance respectively. In the last step, (20), we update the weights. To avoid 0-division a small error term, ε, is set greater than 0.

A big learning rate means that each time the weights are changed, the bigger change is made in contrary to a smaller learning rate. The learning rate is fixed in Adam but step size scales with the calculations of the mean and variances moving averages.

Activation functions

Two activation functions are the Relu and Softmax functions. The Relu (Rectified Linear Unit) activation function introduces non-linearity into the net described in (21).

$$f(x) = \max(0, x) \tag{21}$$

The Softmax activation function is common in the output layer since it outputs values between 0 and 1with a total sum of 1. The output could then be considered a probability distribution over the classes. The Softmax function is described in (22).

$$Softmax(x_k | x_1, x_2, \dots, x_n) = \frac{e^{x_k}}{\sum_{i=1}^{n} e^{x_i}} \tag{21}$$

Loss functions

The loss function, also known as error or objective function, is a measuring method of how big the error in the model's prediction there is with respect to the ground truth [10]. The loss function used in this work is the categorical cross-entropy. The categorical cross-entropy consists of a Softmax activation function followed by a cross entropy calculation.

The cross-entropy is presented in (22) where p is the true probability distribution and q is the predicted probability distribution.

$$H(p,q) = -\sum_{i} p_i log_2(q_i) \tag{22}$$

Since our datasets have a ground truth of either 0 or 1, the cross-entropy calculation simplifies to one non-zero element in the summation. The Softmax is used as an activation function in the classification layer. The Softmax function, which is described in more detail below, normalizes the output and makes it into a probability distribution. Each value in the output vector ranges from 0 to 1. If the output vector from the classification layer is $O = [5,7,12,6,10]^T$, then the corresponding Softmax activation output is:

$$Softmax\left(\begin{bmatrix}5\\7\\12\\6\\10\end{bmatrix}\right) = \frac{e^{x_k}}{\sum_{i=1}^{n} e^{x_i}} = \begin{bmatrix}0.0008\\0.0059\\0.8730\\0.0022\\0.1181\end{bmatrix}$$

Given that the true probability $q = [0,0,1,0,0]^T$, the corresponding cross entropy loss will be:

$$CE(Softmax(O), q) = CE\left(\begin{bmatrix}0.0008\\0.0059\\0.8730\\0.0022\\0.1181\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\\0\end{bmatrix}\right) =$$

$$(log_2 0.0008) \times 0 + (log_2 0.0.0059) \times 0 + (log_2 0.8730) \times 0 + (log_2 0.8730) \times 1 + (log_2 0.0022) \times 0 + (log_2 0.1181) \times 0 = \underline{0.0590}$$

If $q = [1,0,0,0,0]^T$, the corresponding cross entropy loss would instead be:

$$(log_2 0.0008) \times 1 + (log_2 0.0.0059) \times 0 + (log_2 0.8730) \times 0 + (log_2 0.8730) \times 0 + (log_2 0.0022) \times 0 + (log_2 0.1181) \times 0 = \underline{3.0990}$$

As shown, a high value in the classification neuron corresponding to the true value gives a low loss and vice versa.

2.2.3 Regularization

The regularization method used in this thesis is dropout. Dropout turns of neurons with a probability X on specific layers. With "turn off" meaning setting the output value to 0 [11]. This makes the network independent of some neurons for a given class, with the purpose of getting better to generalize over all neurons in a layer. What happens is that the net gets forced to learn alternative pathways and not depending on specific ones. Figure 5 shows how dropout affects the net with some of the neurons turned off.

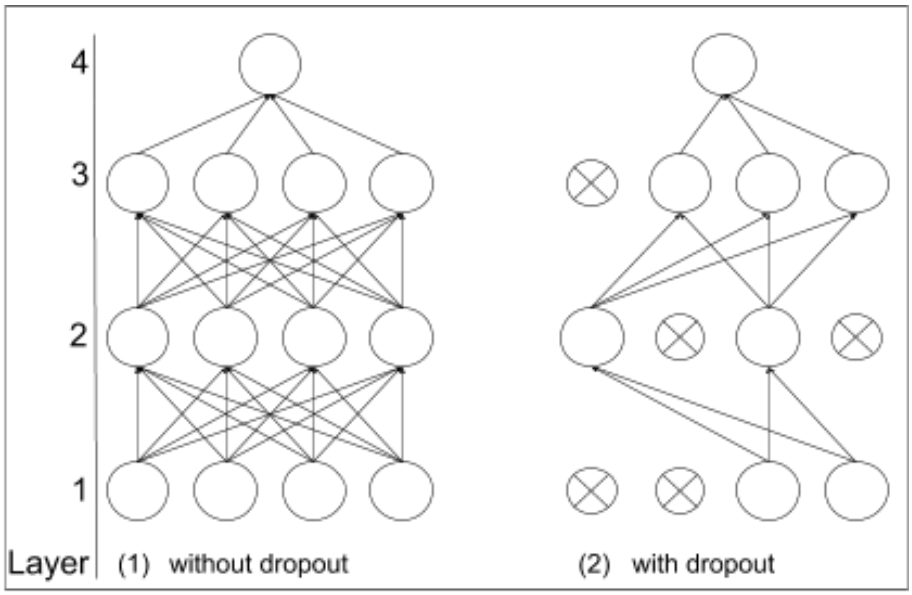


Figure 5: Dropout applied to a neural network with neurons being turned of with a given probability.

2.3 Convolutional neural networks (CNN)

This section starts with a brief description of the VGG16 which is the basic model on which the studied nets are created. Also, the customized nets are described and what differences are made when training on the different datasets. In the next section, the layers in the nets are described. Lastly, the training part and regularization techniques are presented.

One kind of machine learning model is the convolutional neural network (CNN) which is popular in image recognition. For a fixed number of layers, a CNN is less complex in comparison to a FNN in terms of parameters. In addition to the input, hidden and output layers, it also consists of convolutional and pooling layers which are described in section 2.3.1. The CNN preserve spatial information like in the information in images which makes them suitable for a specific type of problems such as the image field.

Simonyan and Zisserman [12] are the main creators of VGG16 and other VGGnets convolutional nets. The models were top contenders in the Imagenet Large Scale Visual Recognition Challenge 2014 [13]. The ILSVRC is an annual competition in which researchers from all over the world compete on including which algorithms classify images best. VGG16 consists of five blocks and three dense layers. Each block contains three convolutional layers and a pooling layer. Each convolutional layer has a filter size of 3*3. The first two dense layers are 4096 elements/ neurons long and the last layer is the classification layer with only 1000 elements. The filter size is 3*3 for all convolutional layers. For our purposes, there were some changes to the original net due to two factors. The first being computational speed. The other factor is that the studied datasets are relatively small compared to what VGG16 is trained for since it was intended for larger images. The original net and the adjustments are presented in the table 2.

Table 2: Changes to the original VGG16-net.

VGG16 net	VGG-MNIST/CIFAR10-net
Input shape 224*224*3	Input shape 32*32*3
Convolution layer - 64 filters	
Convolution layer - 64 filters	
Max Pooling layer	
Convolution layer - 128 filters	
Convolution layer - 128 filters	
Max Pooling layer	
Convolutional layer - 256 filters	
Convolutional layer - 256 filters	
Convolutional layer - 256 filters	
Max Pooling layer	
Convolutional layer - 512 filters	Dense layer - 512 neurons
Convolutional layer - 512 filters	Dropout 50%
Convolutional layer - 512 filters	**Output Dense layer - 10 neurons**
Max Pooling layer	
Convolutional layer - 512 filters	
Convolutional layer - 512 filters	
Convolutional layer - 512 filters	
Max Pooling layer	
Dense layer - 4096 neurons	
Dropout 50%	
Dense layer - 4096 neurons	
Dropout 50%	
Output Dense layer - 1000 neurons	

2.3.1 Convolutional layers

The convolutional layers are especially suited in the image field of machine learning since they preserve spatial relations in the input. CNN's also reduces the amount of computation and parameters compared with if one would choose to use a more conventional type of layer, like feed-forward nets [9]. The CNN parameters are the learning filters or kernels. In figure 6 an input matrix of size 6*6 and a filter of size 2*2 is described. The arrow describes direction at which the filter moves and produces the convolved matrix.

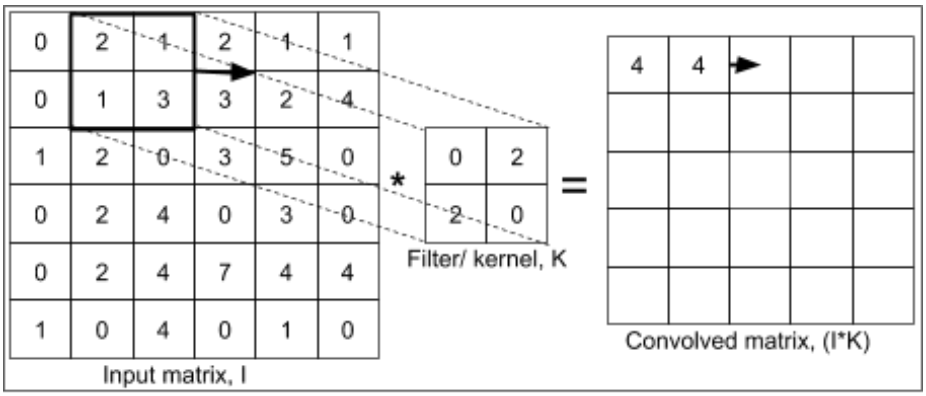


Figure 6: Convolutional process

The convolutional process between an input matrix I and a convolutional filter K in two dimensions is presented in (23). The stride is the number of steps in which the window jumps after each element-wise matrix multiplication. The stride is defined both in the vertical as well as in the horizontal direction.

$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{w} K_{ij} * I_{x+i-1,y+j-1} \tag{23}$$

To introduce nonlinearity into the network, the output $(I * K)_{xy}$, is multiplied with a relu activation function [9].

2.3.2 Convolutional layers

The main function of the max-pooling layer is to reduce the amount of dimensionality of the information, and thus reducing the number of parameters. It is similar to the convolutional layers but instead of doing multiplication or other mathematical operation, we just take the max value for each window. Max pooling is described in figure 7.

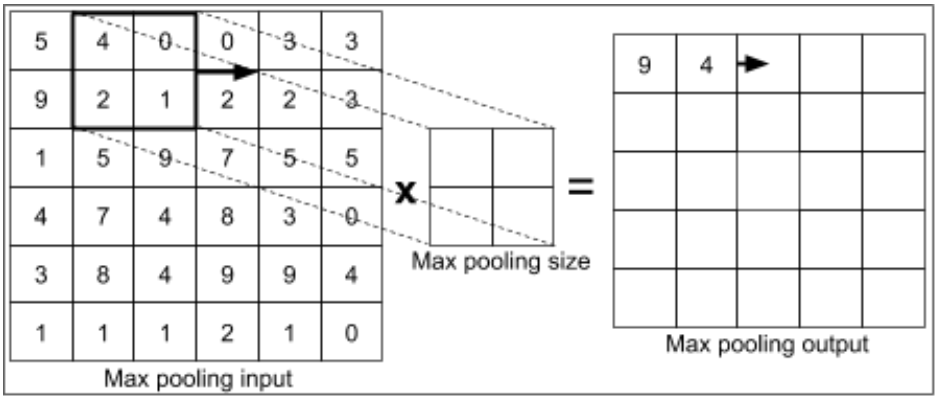


Figure 7: Max-pooling process

The max-pooling process is destructive in terms of preserving information. One can add empty padding columns and rows instead of not pooling the last elements in each dimension [9] if the stride size and input are inconsistent, which figure 8 aims to show.

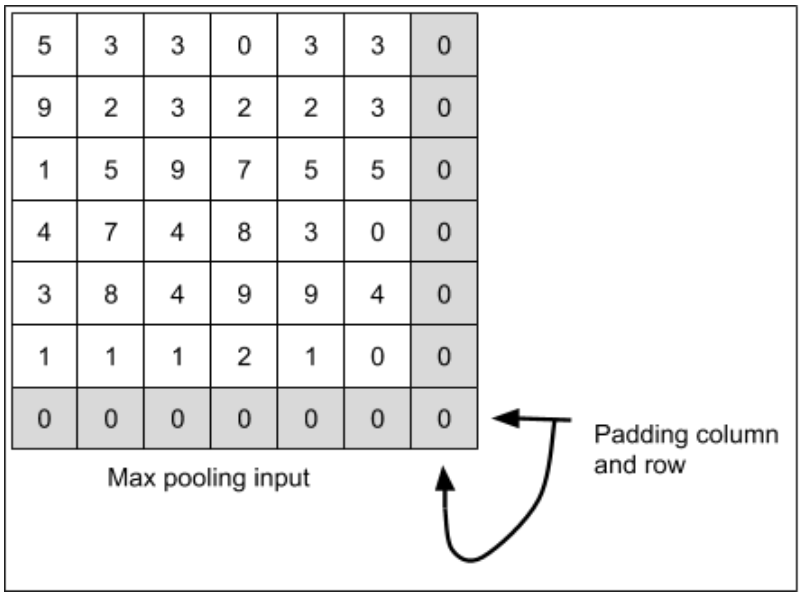


Figure 8: Adding padding row and column

2.4 Time-series

Time series is a sequence of information ordered in time. There are many examples of time series such as stock market price, weather information or heartbeat rate. The loss value of a training instance is the kind of time series this thesis is studying in particular. The time aspect and data points are the epochs and the loss value of the neural network training. Aghabozorgi et al [3] writes "While each time-series is consisting of a large number of data points it can also be seen as a single object". The authors refer to the specific patterns or trends of the time-series.

2.4.1 Preprocessing with standardization and normalization

When preprocessing time-series, it is common to detrend or scale the data. In this thesis, only scaling was studied. Since we are looking for trends and features of time-series rather than the individual magnitude of specific points, we both normalized and standardized the time-series. Figure 9 visualizes the difference between the original, normalized and standardized curves. When normalizing a time-series, the data points either gets squeezed or expanded between the value 0 and 1. Standardizing time-series, on the other hand, create a time-series with a mean of 0 and a standard deviation of 1. Equation (24) and (25) describes the process in mathematical terms.

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{24}$$

$$X_{std} = \frac{X - \mu}{\sigma} \tag{25}$$

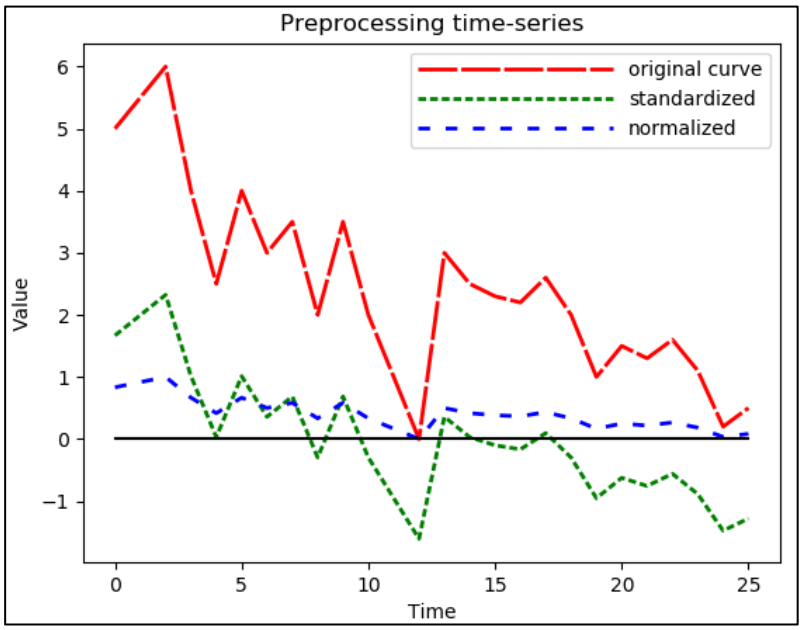


Figure 9: Standardizing and normalizing a time-series.

2.4.2 Distance measuring of time-series

The Minkowski distance, described in (26), between two time-series belongs to a category of distance measuring named lock-step methods [14]. Lock-step methods require that both time-series are of equal length and do a comparison of the point i in time-series x with the point i in time-series y [14]. The method is, relatively to other distance measuring techniques like elastic or feature-based measuring techniques, time-efficient [14].

$$D_{minkowski}(S, Q) = \left(\sum_{i=1}^{n} |s_i - q_i|^p \right)^{\frac{1}{p}} \tag{26}$$

The Minkowski distance is a generalization of the Manhattan distance and the Euclidean distance. When setting p to 1, the distance measured is the Manhattan distance. Setting p to 2 is equal to the Euclidean distance. The Manhattan and Euclidean distances between time-series $S = \{s_1, s_2, s_3, \dots, s_n\}$ and $Q = \{q_1, q_2, q_3, \dots, q_n\}$ is calculated with (27) and (28).

$$D_{manhattan}(S, Q) = \sum_{i=1}^{n} |s_i - q_i| \tag{27}$$

$$D_{euclidean}(S,Q) = \sqrt{\sum_{i=1}^{n}(s_i - q_i)^2} \quad (28)$$

A visualization of the Manhattan distance is described in figure 10, where the size of the arrows corresponds to a big/ small distance in each time-point. The Euclidean distance is hard to visualize for higher dimensions than three.

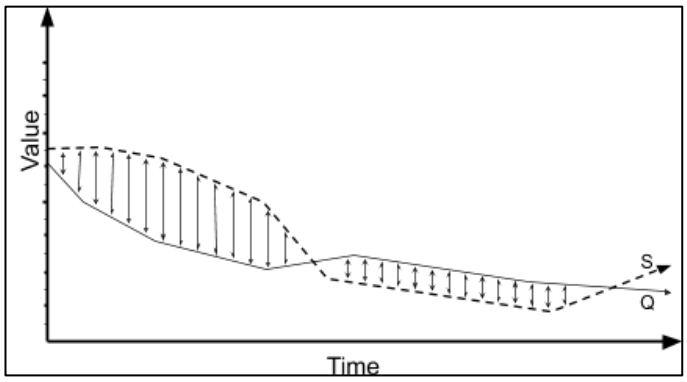


Figure 10: Visualization of Manhattan distance between time-series S and Q.

2.4.3 Mean and median

In many applications when looking for a typical instance of a class, one might assume that the mean of that class is a good representation of it. The problem occurs when there are anomalies in the class. The mean is simply the average of all data points and tends to skew the desired purpose when anomalies are present in the class. Figure 11 aims to show what can happen when the mean and median are used as a representation of a class trend in terms of time-series. The figure is an exaggeration of real-world anomalies, but good for proving the point of what differences one can expect by choosing one or the other. The figure shows the motivation behind the decision to represent class average as median instead of the mean.

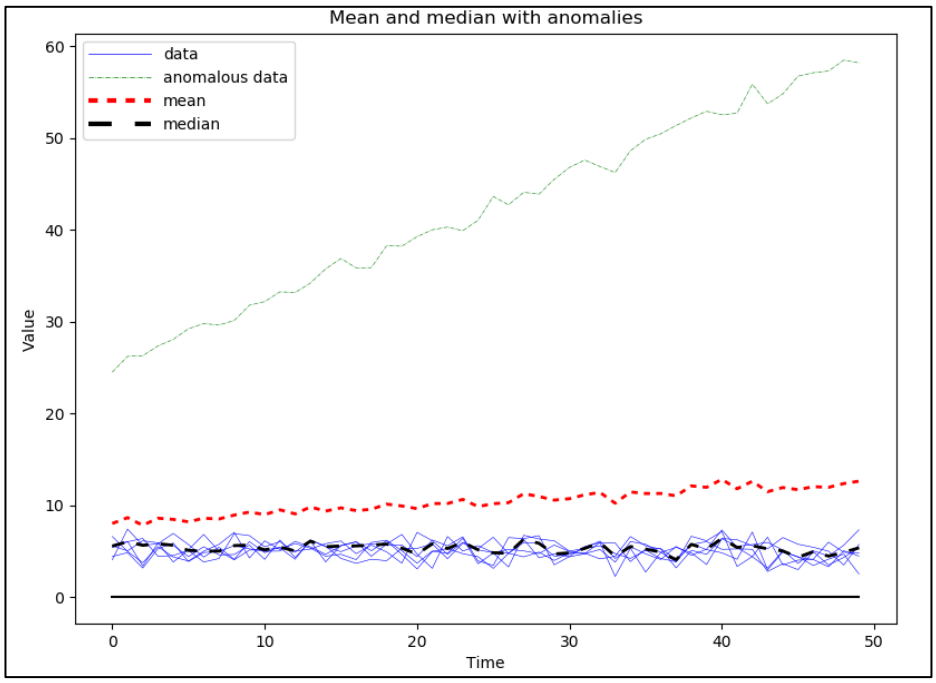


Figure 11: Difference between mean and median when data contain outliers.

2.5 ROC AUC

Accuracy is a common metric to evaluate the score of classifiers. The accuracy is based on calculating the share of correct predictions divided by the number of predictions. Accuracy as a score has some drawbacks, especially in the case of binary classification when the classes are imbalanced. For example, if we have a classifier that predicts whether or not a person has cancer and the classifier always predicts negative, meaning the person doesn't have cancer. Let's say we have 10^1 positive cases and 10^5 negative cases and let the classifier make its predictions. The accuracy of the classifier will be $\frac{10^5}{10^1+10^5} = 0.9999$ which would be misleading in terms of the utility of the classifier. This requires another approach to evaluate the predicting power of the different methods based on two reasons. The accuracy does not handle class imbalances well, as shown in the above example. The second reason is that the proposed AUC is a more nuanced evaluation metric compared to the accuracy [15].

A common way to represent how well a classifier predicts is through a confusion matrix for binary classification described in the table 3. There are four positions for each image and for the perfect classifier the false negatives and false positives are zero, considering no overfitting has occurred.

Table 3: Binary classification categories.

		True classes	
		Positive	Negative
Predicted classes	Positive	True Positives TP	False Positives FP
	Negative	False Negatives FN	True Negatives TN

The metrics used to evaluate the performance of the studied methods are the Receiver Operating Characteristics (ROC) and Area Under Curve (AUC).

The ROC curve is a measure of the true and false positive rate given a particular threshold value, described in (29) and (30) respectively.

$$TPR = \frac{TP}{TP + FN} \tag{29}$$

$$FPR = \frac{FP}{FP + TN} \tag{30}$$

A classifier that predicts randomly is represented in an ROC plot by a diagonal line from bottom left to the top right corner. For a perfect classifier, the line goes through the top left corner. The ROC and AUC are visualized in figure 12. Both ROC and AUC is a relative metric meaning it is good when comparing methods against each other. The metrics do not say much about whether methods are good or bad, as long as they are over the diagonal curve. In this study, we will evaluate a single scalar value of the AUC as a comparing metric between the methods.

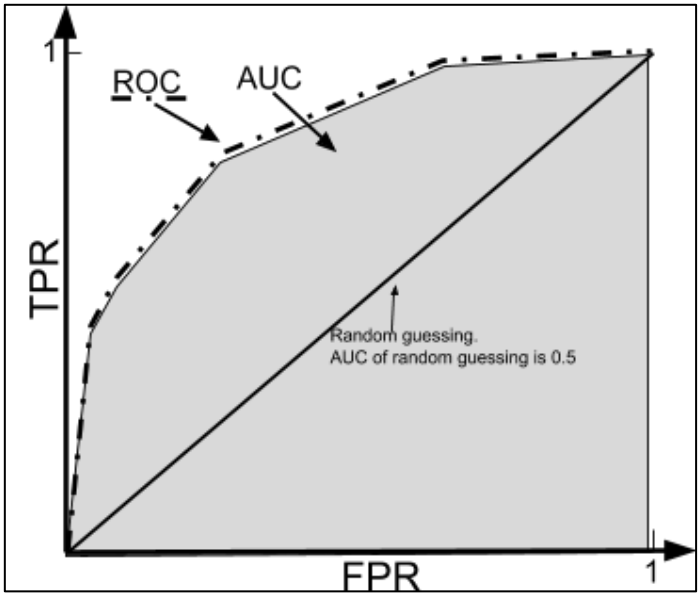


Figure 12: ROC plot. AUC represents the shaded area.

One has to specify a threshold limit when creating a ROC curve. The threshold limit in this study is the true number of errors, which in real-world cases are unknown. Figure 13 shows what impact different values of the threshold makes.

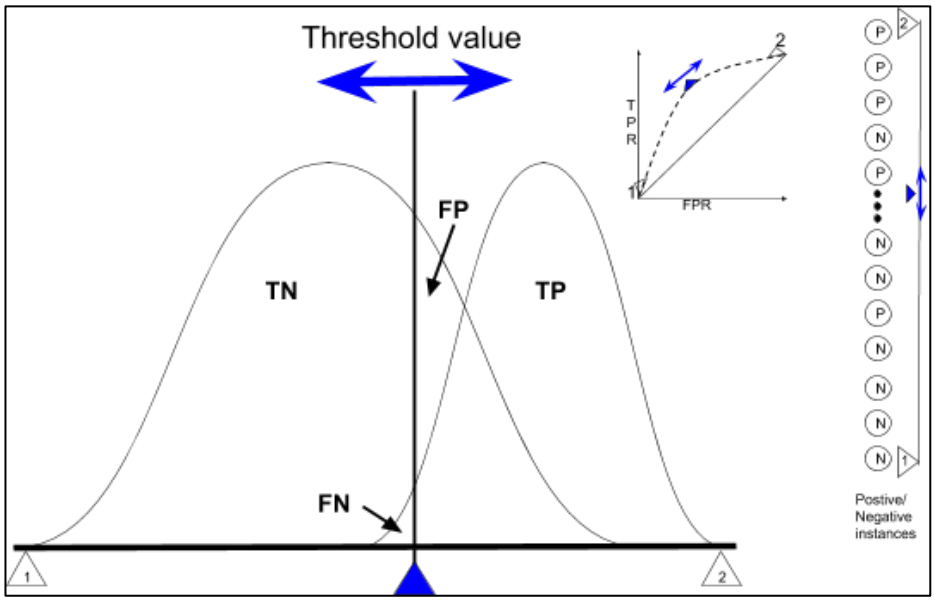


Figure 13: A visualization of what the impact is of different threshold values

The ROC curve can be used to tune the threshold value for a specific purpose. It is important to have a low false-positive count in medical diagnosis. Since it's better to falsely diagnose a negative person as positive than it is to falsely diagnose a positive person as negative. This would correspond to a point closer to 1 than 2 in figure 13. In the discussion chapter, there will be some thought on how to properly choose a limit.

3. Experiment

In this chapter, the experiment will be presented. First, some explanation will be made on the particular software and hardware used. Secondly, the studied dataset will be presented. The methods will be described later. The last two sections contain information about how the experiment was run and what kind of evaluation score was used. One important thing to note is that the amount of errors in the dataset is known during this experiment. In real-world cases, the number would be unknown. Therefore, as also described in the future work chapter, a good practice for picking a threshold limit remains to be studied.

3.1 Software and hardware

The code was implemented in Python 3.6. Several Python libraries were used, most importantly NumPy, SciPy, Keras, and TensorFlow among others. Keras is a high-level library that runs on top of TensorFlow. TensorFlow does all the tensor, multidimensional array, and matrix, computations. The operating system is Linux Ubuntu. Tensorflow and Keras allow running code on the GPU through the CUDA-toolkit library. The GPU is GeForce GTX 1070 and has a RAM size of 8 gigabytes.

3.2 Datasets

In this section, the studied datasets will be described. Both datasets are often used to measure benchmark metrics when creating nets and are regarded as ground truth data by people studying machine learning models [8]. The datasets are standardized in terms of image properties which makes it suitable to study. Furthermore, the datasets are between 60000 and 70000 datapoints big which makes the computational time reasonable within the scope of this study.

3.2.1 MNIST

MNIST (Modified National Institute of Standards and Technology) is a dataset with handwritten digits from zero to nine, 10 classes. There is a total of 70000 images. The size of the images is 28,28 for width and height. The classes in MNIST are balanced with 7000 images in each class. The only type of image augmentation that is done in the dataset is a rotation by 10°, as described in the table 7.

3.2.2 CIFAR10

CIFAR10 (Canadian Institute for Advanced Research) is a ten classes 60000 images dataset. The images are colored and more complex than the MNIST pictures. The classes are balanced with 6000 images each. A small subset of the dataset is presented in figure 14.

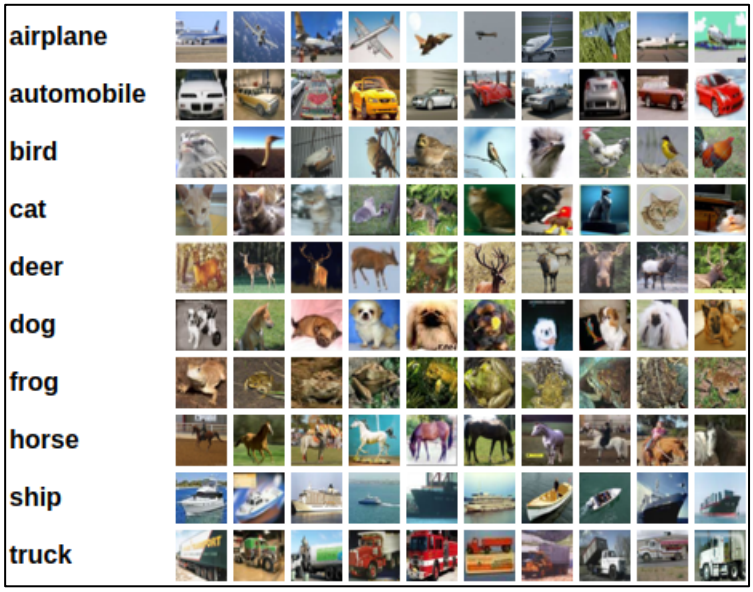


Figure 14: Subset of CIFAR10 dataset [16].

The image size of CIFAR10 is 32,32 for width and height. We make use of image augmentation described in table 7 as we train on the dataset.

3.3 The methods

In this section, four methods will be presented. Two of them are described in part V1 and the other two in part V2. The methods and computations are the same within the parts, with the difference that one of them is normalized and the other is standardized. The presented methods are considered to be a filter, in accordance with Zhang´s [8] idea. In figure 15 the filter is more detailed with the different stages shown.

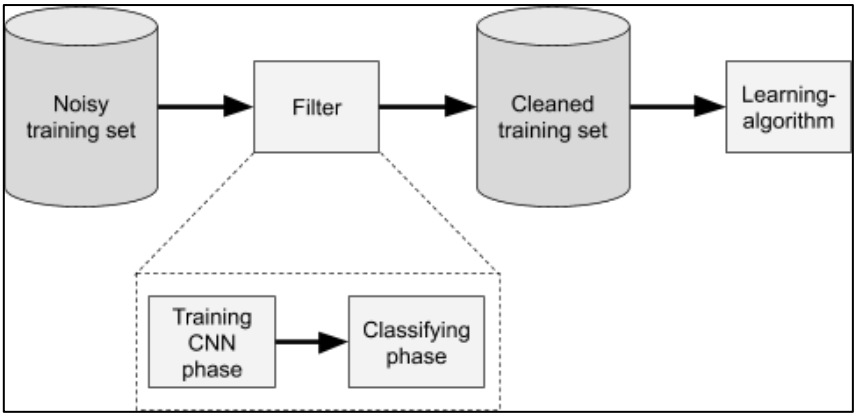


Figure 15: The different filtering stages.

The experiment was made between the epochs 3 and 100. The epochs before is not considered. Comments on the option of starting and ending period will briefly be discussed in the discussion and future work chapters. This section will begin to describe the processes of each method and then an explanatory text will be presented.

V1

The two V1 methods are Euclidean distance-based. One standardizes the training loss of each image while the other normalizes the training loss of each image. The methods will be presented in table 4 with explanatory text below. The input to the method is the loss and label of all images. Image (X, C) means image X with class C.

Table 4: V1-process in parts.

Step	Process
V1,1	Standardize/ normalize all image training losses
V1,2	Calculate median loss for each class and time point
V1,3	Calculate Euclidean distance for each image loss to each median class: $$D_{euclidean}(X, C) = \sqrt{\sum_{i=3}^{100} \left(x_i - c_{median,i}\right)^2}$$
V1,4	Sort the distances in descending order

V2

The V2 methods are Manhattan distance-based. One standardizes the training loss of each image while the other normalizes the training loss of each image. The methods are presented in Table 5 with explanatory text below. The input to the method is the loss and label of all images. Image (X, C) means image X with label class C.

Table 5: V2-process in parts.

Step	Process
V2,1	Standardize/ normalize all image training losses
V2,2	Calculate median loss for each class and time point
V2,3	Calculate Manhattan distance for each image loss to each median class: $$D_{manhattan}(X, C) = \sum_{3}^{100} \left\lvert x_i - c_{median,i} \right\rvert$$
V2,4	Sort the distances in descending order

The first two steps are the same for both methods. In step V1,2 and V2,2, the median is calculated for each image. In step V1,3 and V2,3 the Euclidean and Manhattan distance to label class median is calculated. The output from V1,3 and V2,3 is a one-dimensional array with the distances to the median class loss corresponding to the label class. In the last step, the distances are sorted in descending order.

3.4 Score evaluation

Metrics used for evaluating the performance of different methods are the ROC AUC. In real cases, the number of true errors will not be known. This thesis aims to show if the

methods are considered and what kind of method would be more suitable in identifying erroneous labels when studying losses. Therefore, the threshold when selecting images from the sorted list V1,4 and V2,4 is set to the actual numbers of erroneous images.

3.5 Running the experiment

There were 50 separate runs of training made on the presented VGG-CIFAR10 and VGG-MNIST nets for each dataset at an error level on 1 and 10%. Before training the errors were injected into the training sets while the test was set free from errors. Errors were injected by changing a given label to another label in the dataset. Table 6 shows the algorithmic procedure of creating the cross-fold datasets.

Table 6: description of error injecting in parts.

Step	Process
1	Randomly shuffle the dataset
2	Split the dataset into 10 equally sized sets
3	For i=1,…,10 do:
4	Save the set (before injecting errors)
5	Randomly choose X% of data-points
6	For each chosen data-point:
7	Switch the label to a different label except the given class
8	For i=1,…,10 do:
9	Merge all sets with the injected errors into a new big set
10	Keep the i^{th} set clean from errors (as test set)

10-fold cross-validation was made 5 times on each error level as presented in figure 16. Within each cross-fold validation run, the errors were the same. Between each cross-fold validation run the errors were changed in order for the result to not depend on specific errors. A total of 200 CNN models were trained during this study. The ROC will be presented as an average of all the runs for each method, dataset and error level.

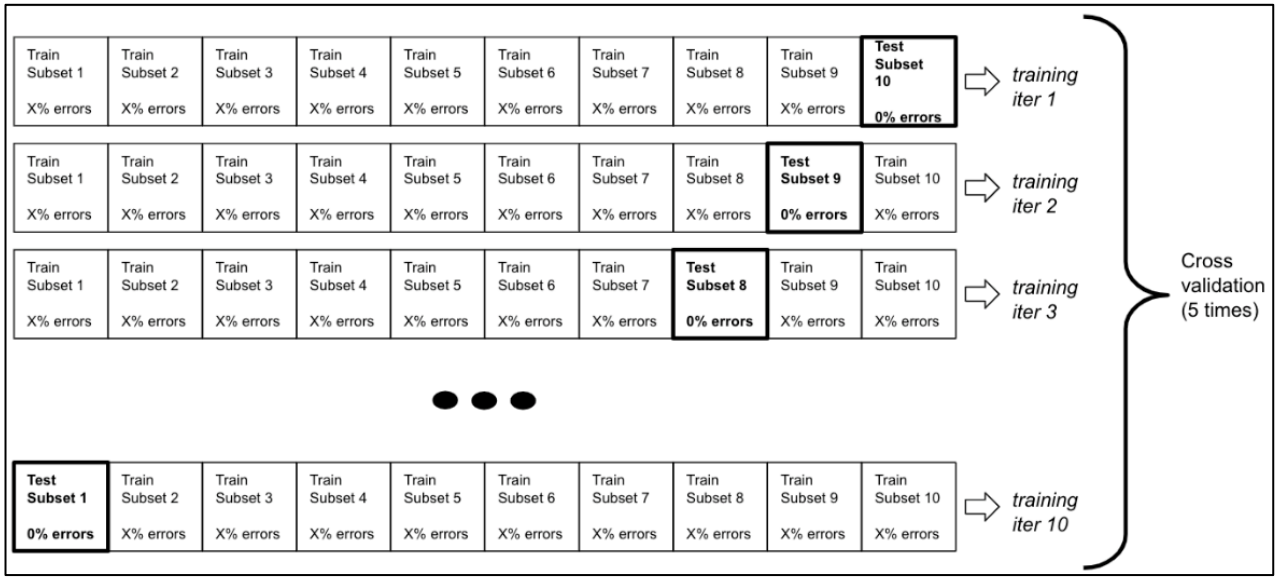


Figure 16: Cross-fold validation setup.

Table 7 describes the image augmentation used for each dataset.

Difference in nets	VGG-MNIST net		VGG-CIFAR10 net	
Image Augmentation	Rotation	10°	Shear	20°
			Rotation	10°
			Width shift	20°
			Height shift	[-20%, +20%]
			Zoom	[-20%, +20%]
			Horizontal flip	[-20%, +20%]

Both nets trained on the datasets for 100 epochs. Both implementations used Adam as the optimization function. No pretrained weights were used. The hyper-parameters were set to; 1e-4, 0.9 and 0.999 for η, β_1 and β_2 respectively. Adam is available in Keras as an optimizing function. One positive aspect of Adam is that large uncertainty, based on the calculation of variance in the momentum, results in smaller steps in terms of minimizing loss function [16].

4. Result

In this chapter, the results from all experiments will be presented. We have run the experiment on each error level and dataset 50 times. Firstly, the experiment results on MNIST will be introduced, and afterward the CIFAR10 results. Since this technique is relatively novel when finding erroneous labels, meta information will also be presented in order to give future researchers more information about the research direction. The reason for presenting the distance histogram is partly to show the distributions of distances and partly to elaborate on how one might choose threshold limits in future implementation. One note on the distance histogram distance plots is the fact that only one experiment is presented for each error level and dataset. The reason for not showing a mean distribution for all experiments is that the AUC standard deviation was small. Furthermore, this study has a fixed threshold limit set to the true number of errors when creating a ROC plot, which will not be known in real case scenarios.

4.1 Running the experiment

A total of 200 CNN's was trained. 50 times on each level and dataset. The training went on during 100 epochs and the results are the mean and standard deviation from all training on each error level and dataset. Figure 17 and 18 shows the results from training the CNN and the ROC of the MNIST dataset with one standard deviation via shading in the plots. The loss and accuracy of both the training and validation part are shown. There was a very low deviation as is shown in the right plot in both figures.

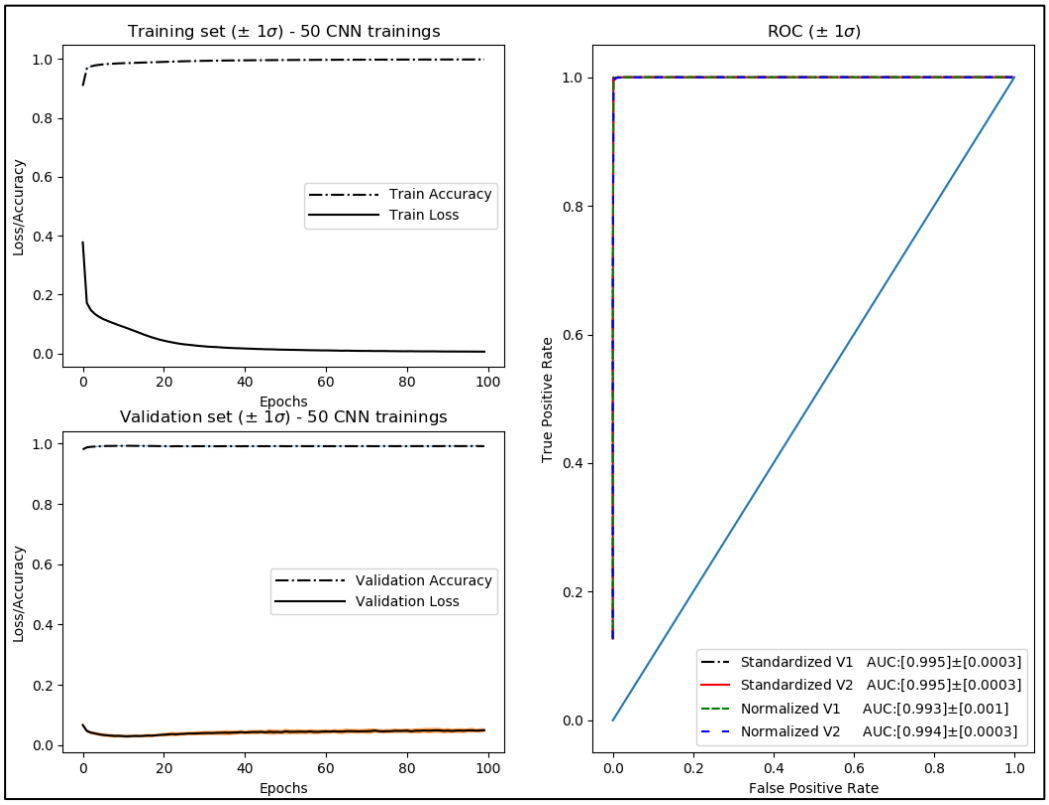


Figure 17: CNN training result and ROC plot for MNIST experiment at error level 1%.

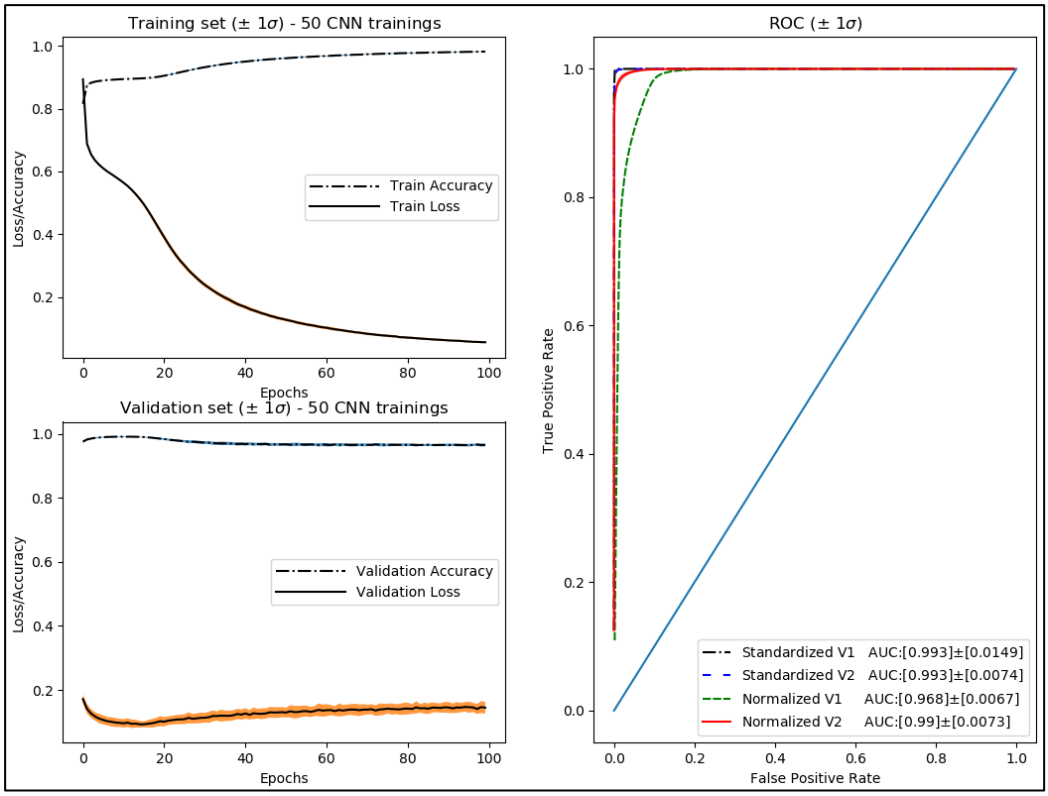


Figure 18: CNN training result and ROC plot for MNIST experiment at error level 10%.

Figure 19 and 20 shows the normalized distribution of the distances per image for one experiment. For a perfect classifying filter, the distribution would have to be totally separated. Also one can see that there are two separate distributions in each plot.

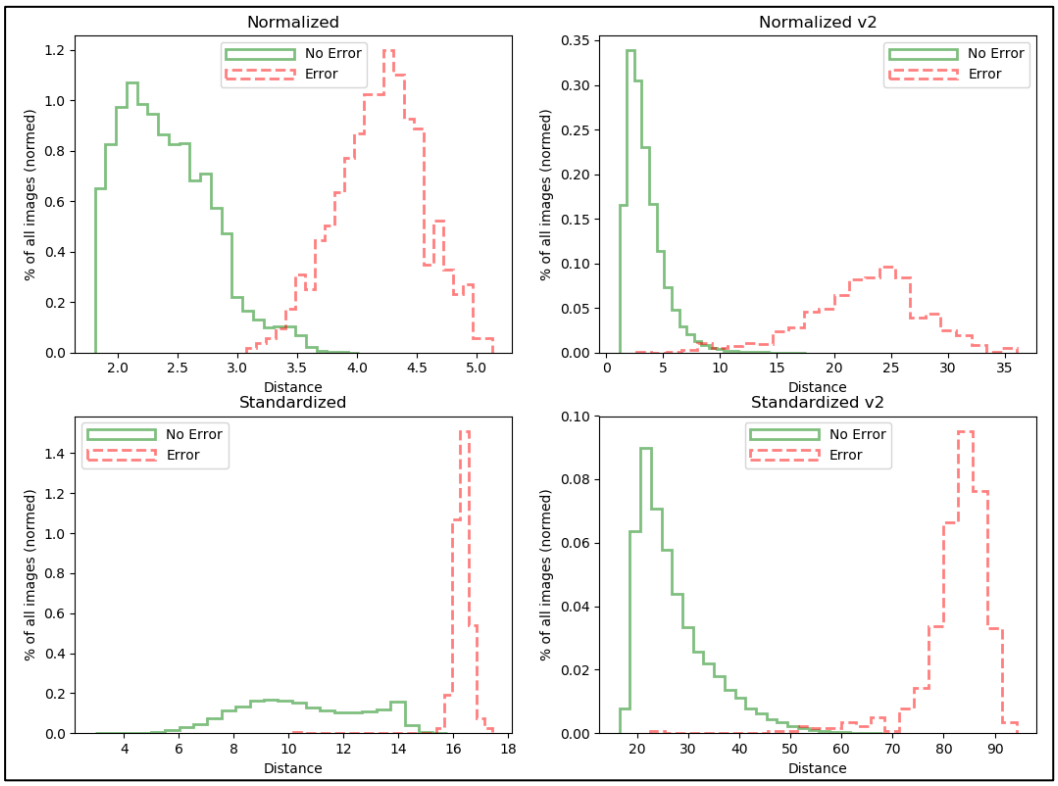


Figure 19: Distribution of the distances for MNIST at error level 1%.

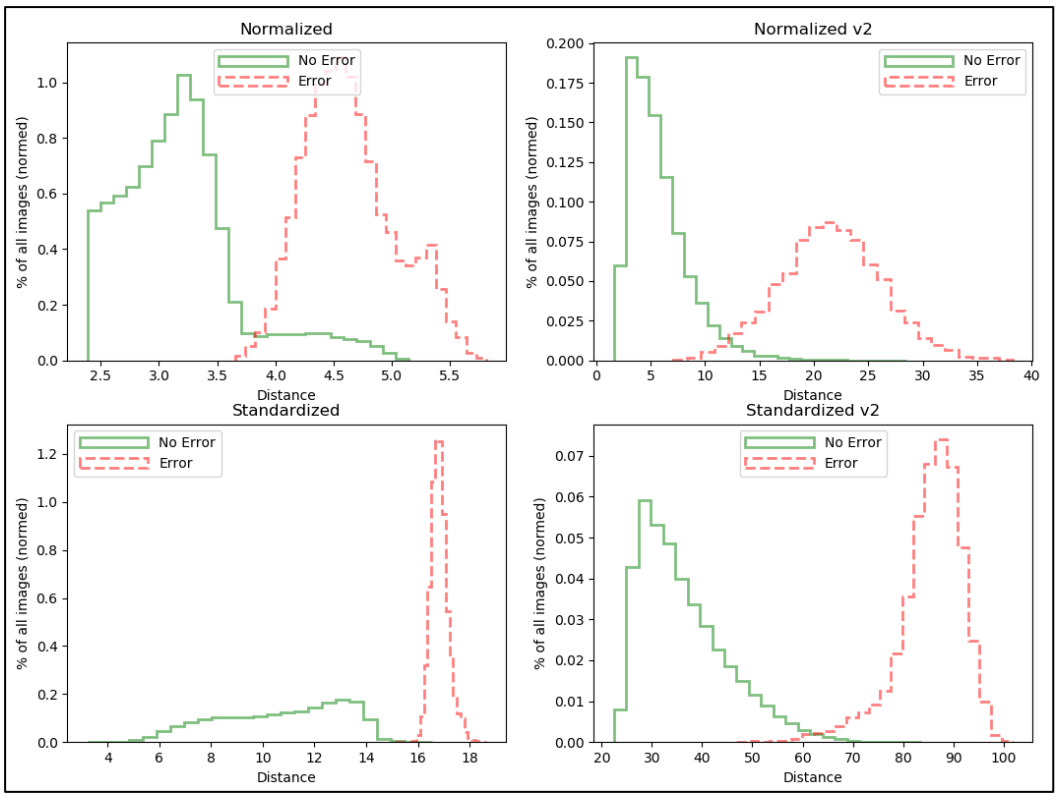


Figure 20: Distribution of the distances for MNIST at error level 10%.

Figure 21 and figure 22 shows how the training and validation accuracy/ loss increases/ decreases for CIFAR10 dataset during training of the CNN and the ROC plots.

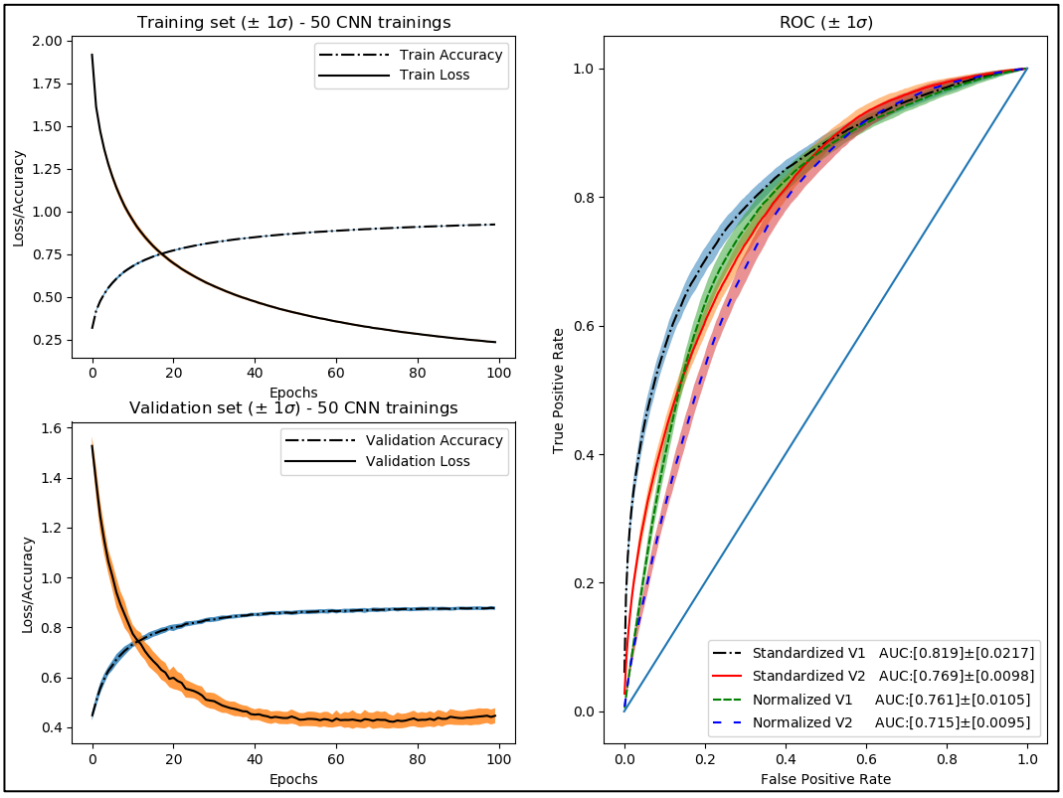


Figure 21: CNN training result and ROC plot for CIFAR10 experiment at error level 1%.

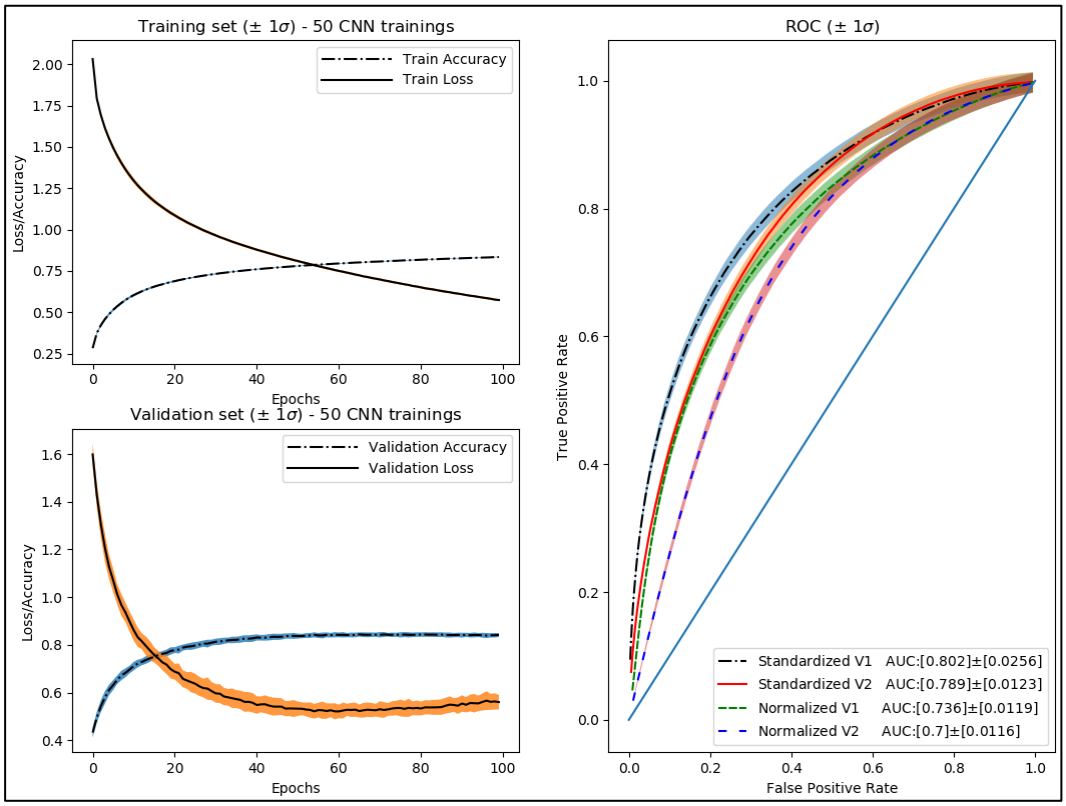


Figure 22: CNN training result and ROC plot for CIFAR10 experiment at error level 10%.

The standard deviation is shown in the shadowed area around the curves. As one might expect, the accuracy is lower in the experiment with 10% errors than in the 1% error dataset. The loss is also lower both for the training and validation set. Both validation accuracies are close to the same value, but the 10% set is slower on achieving high training accuracy than the 1% set. The ROC plot shows that the standardized V1 methods achieve the best classification result on both datasets. The methods that are close to the same results are the normalized V1 and standardized V2 method. In the discussion chapter, there will be presented arguments for why one might be more preferable than the other. The AUC is lower for the dataset with fewer errors. Lastly, one can see that there is a higher uncertainty in terms of AUC in the dataset with higher errors.

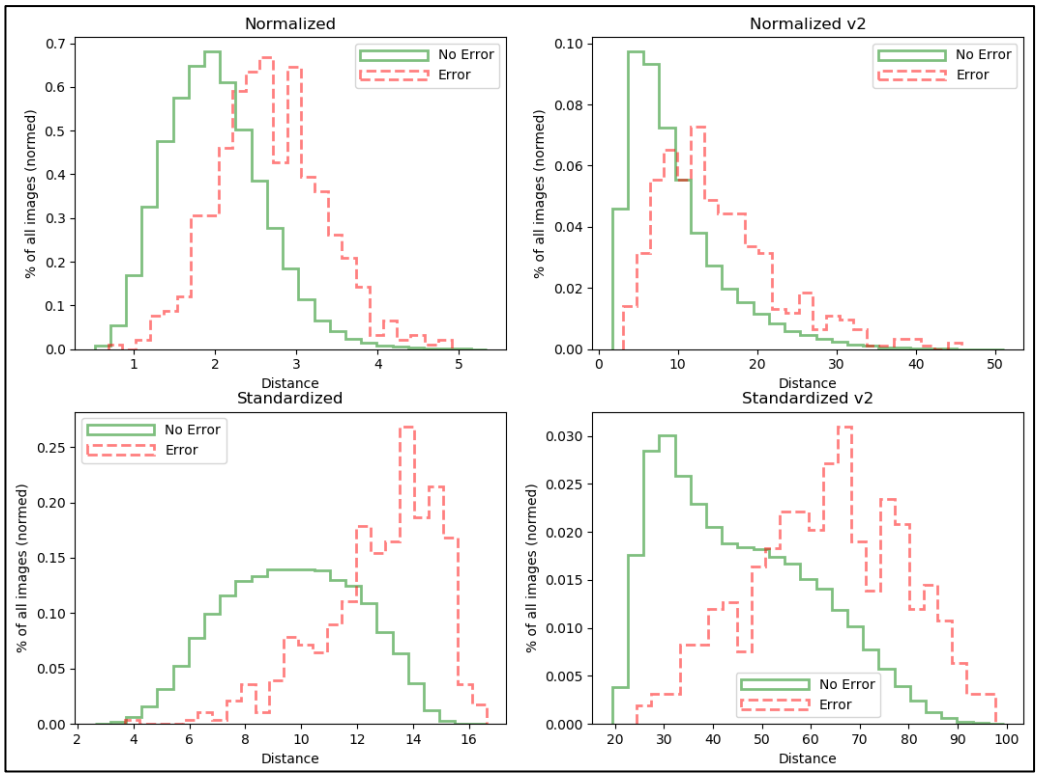


Figure 23: Distribution of the distances for CIFAR10 at error level 1%.

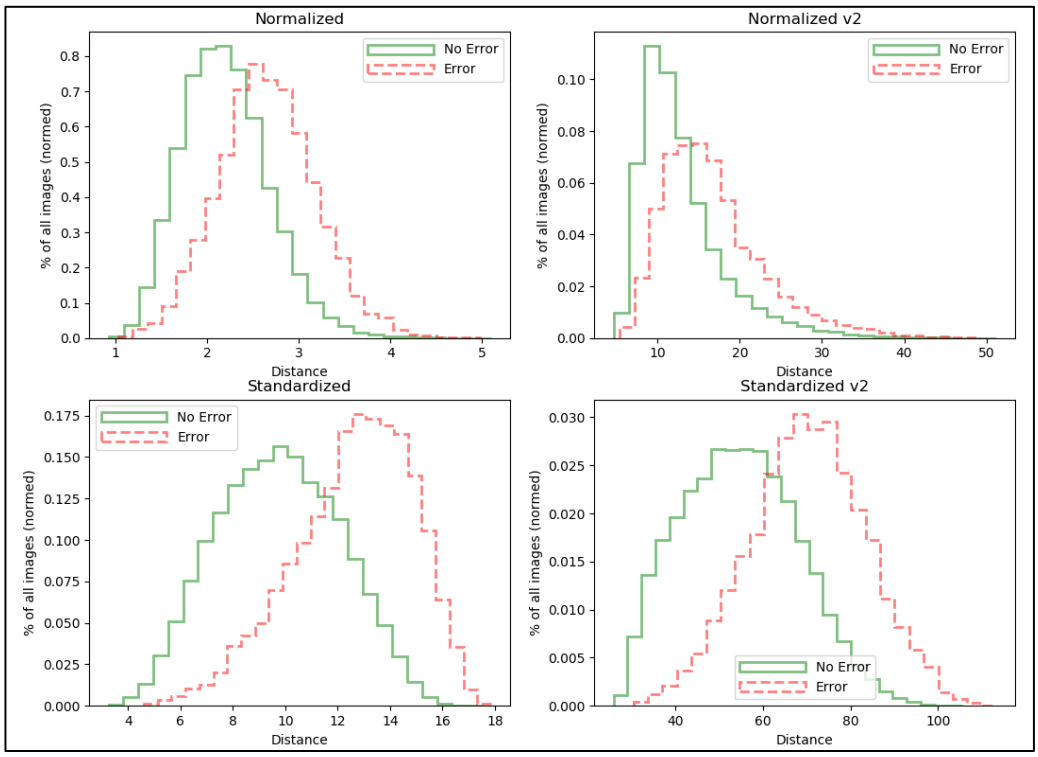


Figure 24: Distribution of the distances for CIFAR10 at error level 10%.

In figure 23 and 24 the distributions are not as separated as in the case of MNIST. Some separation exists, but a lower one, which also shows in the AUC score of CIFAR10 compared to MNIST AUC score.

In table 8 all results are summarized for AUC with 1 standard deviation. Both datasets have the same order in performance for the methods. The standardized V1 method performs best in both cases in terms of AUC.

Table 8: Summary of the result.

		MNIST		CIFAR10	
		AUC μ	**± 1σ**	**AUC μ**	**± 1σ**
1% Error	Standardized V1	**0,9946775**	0,000132	**0,819**	0,0217
	Standardized V2	0,99455844	0,00026745	0,769	0,0098
	Normalized V1	0,99336247	0,00103941	0,761	0,0105
	Normalized V2	0,99437486	0,00030633	0,715	0,0095
10% Error	Standardized V1	**0,99340999**	0,00743853	**0,802**	0,0256
	Standardized V2	0,99286875	0,00743095	0,789	0,0123
	Normalized V1	0,96843707	0,00668214	0,736	0,0119
	Normalized V2	0,98963444	0,00731524	0,700	0,0116

4.2 Answering research questions

Research question 1

The experiment shows that it is possible to find erroneous labels by studying individual losses. All methods show a ROC-curve significantly higher than the random diagonal curve.

Research question 2

To answer which of the methods that perform best, a statistical test is used on the AUC scores. The Wilcoxon signed-rank test is the statistical test and a Bonferroni correction for the family-wise error rate is applied. The Wilcoxon signed-rank test is a non-parametric dependent test. A non-parametric test is done on data which does not have a structure like normal or exponentially distributed data. The alpha level is set to 0.05. After correction, it is rounded and set to 0.01. The critical value is found in a table for one-sided tests. Since the one-sided test is made, one can see if differences in performance between the two methods are statistically significant. When the two-sided test is made, one can only see that there is a difference. The critical value, T_0, which has to be higher than the test score which is 398 in order to be able to reject the H_o. The result is summarized in table 9.

- H_0: There is no difference in AUC score between methods
- H_R: One of the methods perform better than the other one in terms of AUC score

Table 9: Summary of Wilcoxon signed-rank test.

MNIST test cases - 1 % error rate				
case	better method	worse method	T score	H_o rejection
13	standardized V1	standardized V2	0.5	yes
14	standardized V1	normalized V1	0	yes
15	standardized V1	normalized V2	3	yes
16	standardized V2	normalized V1	0	yes
17	standardized V2	normalized V2	0	yes
18	normalized V1	normalized V2	42	yes
MNIST test cases - 1 0% error rate				
case	better method	worse method	T score	H_o rejection
19	standardized V1	standardized V2	0	yes
20	standardized V1	normalized V1	0	yes
21	standardized V1	normalized V2	0	yes
22	standardized V2	normalized V1	0	yes
23	standardized V2	normalized V2	0	yes
24	normalized V1	normalized V2	0	yes
CIFAR10 test cases - 1 % error rate				
case	better method	worse method	T score	H_o rejection
1	standardized V1	standardized V2	0	yes
2	standardized V1	normalized V1	0	yes
3	standardized V1	normalized V2	0	yes

4	standardized V2	normalized V1	12	yes
5	standardized V2	normalized V2	0	yes
6	normalized V2	normalized V1	0	yes
CIFAR10 test cases - 10% error rate				
case	better method	worse method	T score	H_o rejection
7	standardized V1	standardized V2	0	yes
8	standardized V1	normalized V1	0	yes
9	standardized V1	normalized V2	0	yes
10	standardized V2	normalized V1	0	yes
11	standardized V2	normalized V2	0	yes
12	normalized V2	normalized V1	0	yes

The statistical test shows that there is a significant difference between all methods. It shows that the resulting AUC for the methods is almost the same for all experiments except the normalized method. In MNIST experiment the normalized V2 method showed a higher AUC mean for both error levels.

5. Discussion

The results show that high training and validation accuracy is obtained in both datasets and for both error levels. Two possible reasons are that the net, although it is a simplified version of the original VGG-net, is sufficiently complex to achieve high accuracy. The other possible reason is that the datasets are relatively simple and do not have to train for 100 epochs. There are some indications on overfitting in the training of the CNN. These indications are increasing validation loss while the validation accuracy stays stationary. It's unclear whether less overfitting, i.e. stopping the training earlier, will affect the score. The causal relationship is something that is not studied in this thesis but is mentioned in future work. Furthermore, since the main purpose of the CNN training is not to train a CNN that is good at generalizing, but rather a method to find erroneously labeled data. Therefore, the overfitting of training the CNN is not necessarily a concern as if the goal was to train a CNN classifier. The discussion point is that it could be the case that some overfitting is needed for the method to find erroneously labeled data. Furthermore, the validation accuracy on CIFAR10 is lower, which is expected since it’s a more complex dataset.

The results also show that it is possible to find erroneous images by studying the loss of each individual image. Especially with the MNIST dataset and the applied methods, one can find errors. An AUC of nearly 1 with the MNIST dataset can be considered good and the methods are applicable, which is reflected in the histograms of distances. The two distributions of errors and non-error sets are clearly separated. For CIFAR10, there is some separation, but the two sets are merged in higher degrees. With a more complex dataset, one has to decide how to pick a good threshold rule. A possible approach is drawing 1% of images from a dataset and then manually label them in order to approximate what threshold level to set. If a dataset consists of 100 000 images. Randomly picking 1000 images (1%) would give some information about the error level in the dataset. One might have different sources labeling the same 1000 images to minimize human errors and thus get a more correct error level estimation.

In real world datasets, the errors in dataset can be dependent on the class. This thesis has made random label switches when injecting errors. For instance, a cat can have a higher probability to wrongly be labeled as dog than as a frog. A dependent error injection is a subject for future work.

We can see in the statistical tests that there is a clear ranking of method performance for each dataset. According to the statistical test, the standardized V1 method outperforms all other methods. The H_o is rejected in all tests. That also indicates that there is a significant hierarchy in the performance of the methods for each dataset.

6. Conclusion

We can conclude that it is possible to extract erroneously labeled images from the dataset by studying the loss function for each individual image. Both the distance histograms and the AUC scores support the conclusion.

There is a clear hierarchy in the performance of the method according to the statistical test performed. The result suggests that standardizing the loss function and applying the V1 method which is a Euclidean-based distance measuring method a potential approach.

7. Future work

This thesis has shown that it is possible to find erroneous labeled data when studying loss for all images. I will end this thesis by leaving a couple of potential research directions.

First of all, the most important future work would, in my opinion, be concerning finding rules when setting threshold value. It would also be important in terms of what kind of errors one might accept. As I presented in the discussion part, sometimes it is more important to have a low count of false positives, as in the case of medical diagnosis, than a high count of false positives. An alternative approach is presented in the discussion chapter, which is randomly selecting 1% of the dataset and manually label it. An approximate value would then be obtained from the error level in the dataset.

It would be interesting to vary the number of epochs and see how it would impact the score. This thesis has only studied from epoch 3 to 100. Therefore, in future work, one might vary the number of epochs based on some indicators and see if there is a preferable rule.

It would also be interesting to see if other types of distance measuring methods would give better results. This thesis only includes a Euclidean distance. Other potential distance measuring techniques are elastic measuring techniques like dynamic time warping or feature-based techniques like discrete Fourier transformations.

Lastly, it would be interesting to inject errors dependent on the class rather than in random.

8. Acknowledgements

I want to start by thanking Teorem AB and Emil Romanus for making this thesis possible and for all the support. Thanks to my supervisor Johan Nyberg for good discussions. I would also like to thank you for helping me with the thesis and statistics in particular. Thanks to Daniel Hjerth with code implementation and machine learning theory. Im thankful that you took the time to improve my programming skills.

I want to thank my subject reviewer Michael Ashcroft for all the constructive discussions and comments. I would also like to thank you for helping me with thesis structure and theory. Lastly, thank you for showing such an interest in my work which made me more motivated.

References

[1] E. Dumbill, "Making Sense of Big Data," *Big Data,* vol. 1, pp. 1-2, 2013.

[2] Gartner, "Newsroom, Press Release," 16 9 2015. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2015-09-16-gartner-survey-shows-more-than-75-percent-of-companies-are-investing-or-planning-to-invest-in-big-data-in-the-next-two-years. [Accessed 01 05 2019].

[3] S. Aghabozorgi, A. S. Shirkhorshidi and T. Y. Wah, "Time-series clustering - A decade review," *Information Systems,* vol. 53, 2015.

[4] B. Zerhari, "Class Noise elimination approach for large datasets based on a combination of classifiers," in *2nd International Conference on Cloud Computing Technologies and Applications*, Marrakech, 2016.

[5] B. Frénay and M. Verleysen, "Classification in the presence of label noise: A survey," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 25, pp. 845-869, 2014.

[6] C. E. Brodley and M. A. Friedl, "Identifying Mislabeled Training Data," *Journal Of Artificial Intelligence Research,* vol. 11, pp. 131-167, 1999.

[7] X. Zhu, X. Wu and C. Qijun, "Eliminating Class Noise in Large Datasets," in *Proc. of the Twentieth International Conference on International Conference on Machine Learning*, Washington, 2003.

[8] X. Zhang, "An Improved Method of Identifying Mislabeled Data and the Mislabeled Data in MNIST and CIFAR-10 Appendix Findings in Fashion-MNIST," *SSRN Electronics Journal,* vol. 1, 2018.

[9] K. O'shea and R. Nash, "An Introduction to Convolutional Neural Networks," *CoRR,* vol. abs/1511.08458, 2015.

[10] D. P. Kingma and J. Ba, "A method for stochastic optimization.," *CoRR,* vol. abs/1412.6980, 2014.

[11] F. Chollet, Deep Learning with Python, vol. 1, Greenwich, CT: Manning Publications CO., 2017.

[12] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

[13] A. Zisserman and K. Simonyan, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *CoRR,* vol. abs/1409.1556, 2014.

[14] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV),* vol. 115, pp. 221-252, 2015.

[15] P. Roelofsen, "Time series clustering," Vrije Universiteit Amsterdam, Amsterdam, 2018.

[16] C. X. Ling, J. Huang and H. Zhang, "AUC: a Statistically Consistent and more Discriminating Measure than Accuracy," in *Proc. 18th International Joint Conf. Artificial Intelligence (IJCAI)*, Acapulco, 2003.

[17] A. Krizhevsky, "CIFAR10," 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html.