Course:

# Predictive and Descriptive Learning

Final Deep-learning Report

---

Name: (Olle Elvland)

Date: 2023-01-10

# 1. INTRODUCTION TO DEEP-LEARNING PROJECT

## 1.1. Problem description

Deep learning is a subset of machine learning that uses deep neural networks, which are neural networks with multiple layers, to model and solve complex problems. These networks are trained on large datasets and are able to extract features, train models and make predictions on complex data with high accuracy and generalization capabilities. These networks are widely used in industries such as computer vision, speech recognition, and natural language processing. They achieve state-of-the-art results in many problem domains, and are characterized by their ability to automatically learn features and representations in the data with increasing complexity through the layers. In short, deep learning is a technique that uses deep neural networks to improve the ability to analyze and make decisions on complex data by automatically learning features and representations through multiple layers.

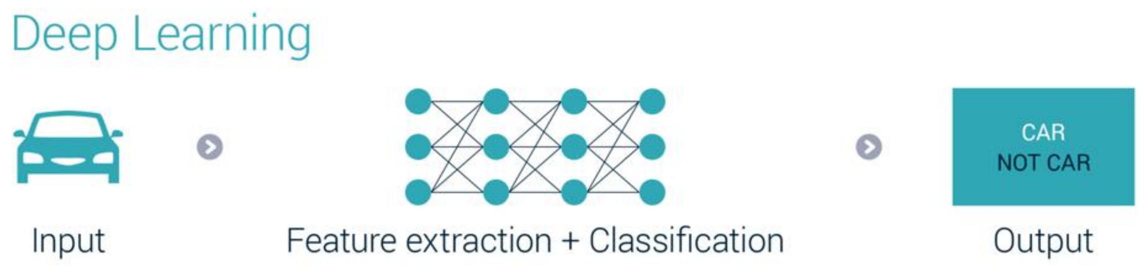


**Figure 1 Deep learning illustration**

In this final project in the PRDL course we were assigned an assignment of using and learning about basic DL models (such as Feed forward, CNN and RNN; also basic knowledge and understanding about Attention and transformers) understanding the relevance of and the importance of using data augmentations for training and learning strategies (like dropout, batch normalization techniques). What transfer learning is and why fine tuning can be useful and efficient for. We were told to be creative by designing our own simple experiment by our own choice.

# 2. My chosen project(s), tools & libraries

Since I had a real difficult time deciding what kind of project to choose, I decided to implement two small simple supervised mini projects based on the basic DL models to maximize my understanding within neural networks. Each with different kind of deep-learning architectures and task corresponding to each network. The two dissimilar projects consisting of:

1) A **feed forward neural network** architecture for determining if rain will appear or not based on humidity the day before
2) A **Convolution neural network** to classify different cloth images using the fashion MNIST dataset using well known convolution layers such as convolution, pooling and a fully connected layer and try regularization techniques such as dropout.

One of the most widely used data science programming language for visualizing and conducting data analysis is being used for this deep learning project is *Python*

*Python* is considered as the base for many data scientist. It is believed to be one of the most popular programming languages in the word. The language was first released in 1990 and has been open source ever since. It has become very popular over last year's due to its extremely high uses for all types of programming tasks. Thanks to its simplicity the high-level language is very easy to understand and learn, since most of the syntaxes try to mimic the human language as much as possible. Also, du to its open source availability, has led to all sorts of people have helped the language grow incredibly fast. Pythons possibility are limitless, and the learning curve is said to be fairly low. Thanks to its popularity, hundreds of different useful libraries and frameworks is being ready to be. The growth of interest in recent years within AI, big data, machine learning and data sciene has also been one of the main factors why Python has become one state of art language. Some of the most helpful libraries (which was also used in the assignment ) in python that is tailor suited for a data scientist :

- **Pandas:** Great framework for data analysis and data handling. Great tool for provide data manipulation control like data frames.

- **Numpy:** Library used high-level math functions and numerical computing with data operations.
- **Matplotlib:** data visualization, cross-platform and graphical plotting library for python
- **Scikit-Learn:** Great library for working with complex data

Anaconda is an open-source distribution of the Python languages that aims to simplify package managements and development. The python distribution may be interfered as an environment , package and data collection management.

Python is one of the fast growing programming language and is also considered as one of the Superior data science, because of its rich tools in terms of performing great mathematical and statistical
tools.

The IDE used for the python part of the machine learning laboratory work is the web IDE called **Google Colaboratory.** The environment is 100 percent web based only suited for python. Colab provide excellent tools for data scientist since its main focus is to implement Deep learning and Machine learning projects assisted by cloud storage. Google Colab is a very popular web IDE because of the system's configurations you are working on is not relevant. The computational resources won't matter. This is a very essential for AI projects as it is a very computational heavy field.

Figure 2 Google colab sign

Both mini-projects were implemented using the open-source software library, TensorFlow. The library provides a wide range of tools for building and deploying machine learning models, including support for training and inference, as well as tools for building and deploying deep neural networks. On top of Tensorflow, Keras wass used which is a high-level neural networks API, written in Python and capable of running on top of TensorFlow . Keras is often used for building and training neural networks for image and speech recognition, natural language processing, and other tasks. Therefore, Keras is a perfect use for implanting my two tasks since I will focus a great deal on image recognition.

# 3. Project 1) Feed forward network theory.

### 3.1.1 Feed forward neural network structure

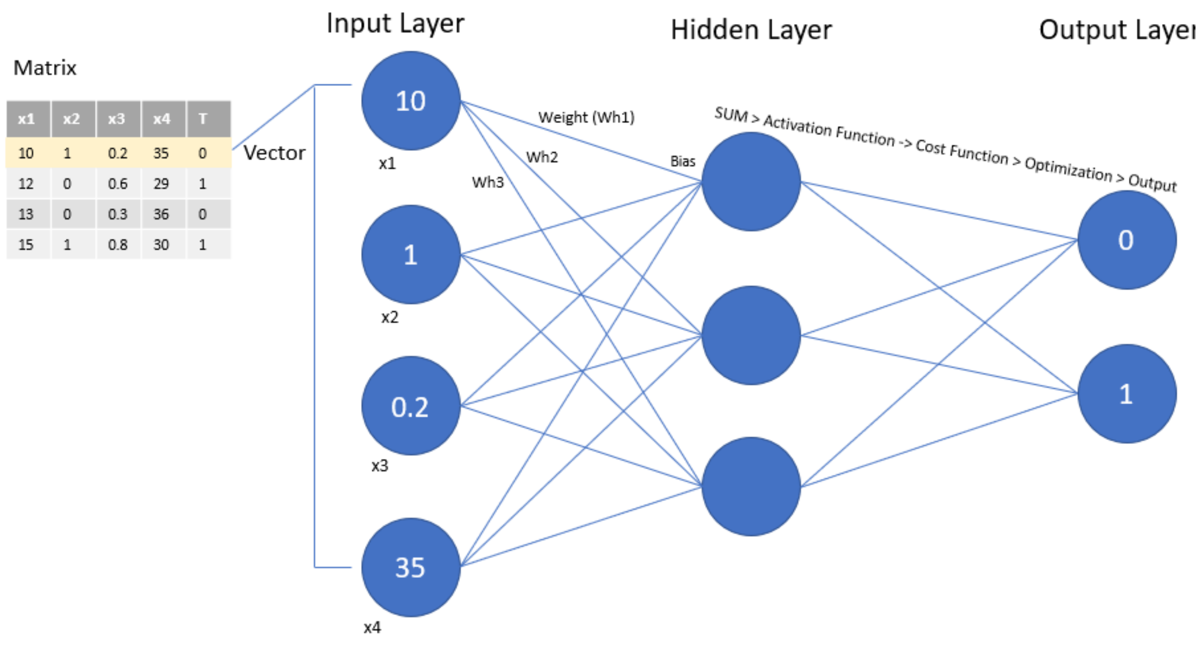
A feedforward neural network is a considered as the most basic artificial neural network in which data flows in one direction from input to output, with no loops or cycles. The structure of a feedforward neural network is composed of several layers, with each layer containing a set of nodes, also known as artificial neurons.

The first layer in a feedforward network is the **input layer**, which receives the input data (normally a vector). The input layer does not typically perform any computations and simply serves as the entry point for the data. Each node in the input layer corresponds to a single input feature.

The second layer is the **hidden layer**, which performs computations on the input data and produces a set of output values that are passed to the next layer. The number of hidden layers and the number of nodes in each layer can vary depending on the specific architecture of the network. Usually, before the data gets propagated to next layer , the data gets transformed using activation functions. The activation functions will results in a finding more complexed patterns and implement non-linearity.

The final layer is the **output layer**, which produces the network's final output. The number of nodes in the output layer corresponds to the number of output values produced by the network.

Between each layer, there are set of **weights** (w1,w2,w3..,w_n) and a bias which are learned during the training process, the most common algorithm is backpropagation, each weight corresponds to the relationship between each input and each node in the next layer.



| x1 | x2 | x3 | x4 | T |
|---|---|---|---|---|
| 10 | 1 | 0.2 | 35 | 0 |
| 12 | 0 | 0.6 | 29 | 1 |
| 13 | 0 | 0.3 | 36 | 0 |
| 15 | 1 | 0.8 | 30 | 1 |

Figure 3 Simple feed forward network with one input layer, one hidden layer and an outputlayer

## 3.1.2 Weights and biases parameters

In a feedforward neural network, the parameters are the weights and biases of the artificial neurons in the network.

- **Weights**: They are the values that control the strength of the connection between two layers, they are the parameters that get learned during the training process. They are matrices of values that are multiplied with the input data to produce the output of each neuron.
- **Biases**: They are the values that are added to the output of each neuron after the weights have been applied. Like weights, biases are also learned during the training process. They are a scalar value that is added to the output of each neuron.

These parameters are learned during the training process, which is typically done using a supervised learning algorithm such as backpropagation. The goal of the training process is to adjust the weights and biases of the network in such a way that the network produces the correct output for a given input.

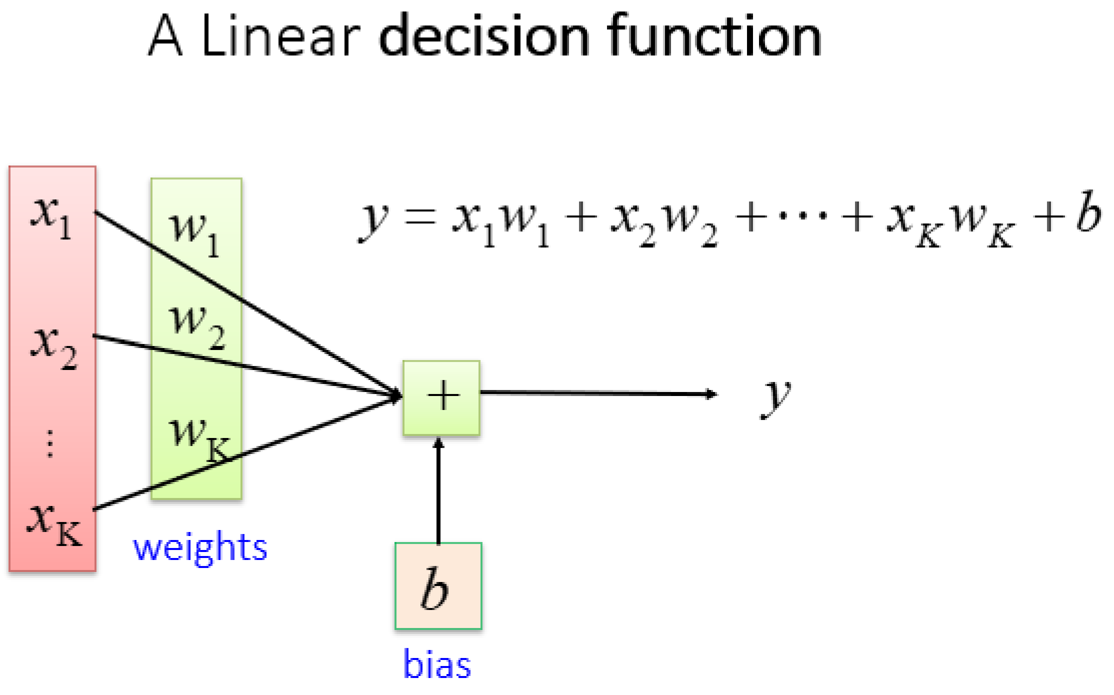

A Linear decision function
$y = x_1 w_1 + x_2 w_2 + \cdots + x_K w_K + b$
$x_1$
$x_2$
$\vdots$
$x_K$
$w_1$
$w_2$
$w_K$
weights
$+$
$y$
$b$
bias

Figure 4 Linear decision function

### 3.1.3 Activation function

An activation function is a mathematical function that is applied to the output of each neuron in a neural network. It serves to introduce non-linearity into the output of the neuron, allowing the neural network to learn more complex and powerful relationships between the input and output data.

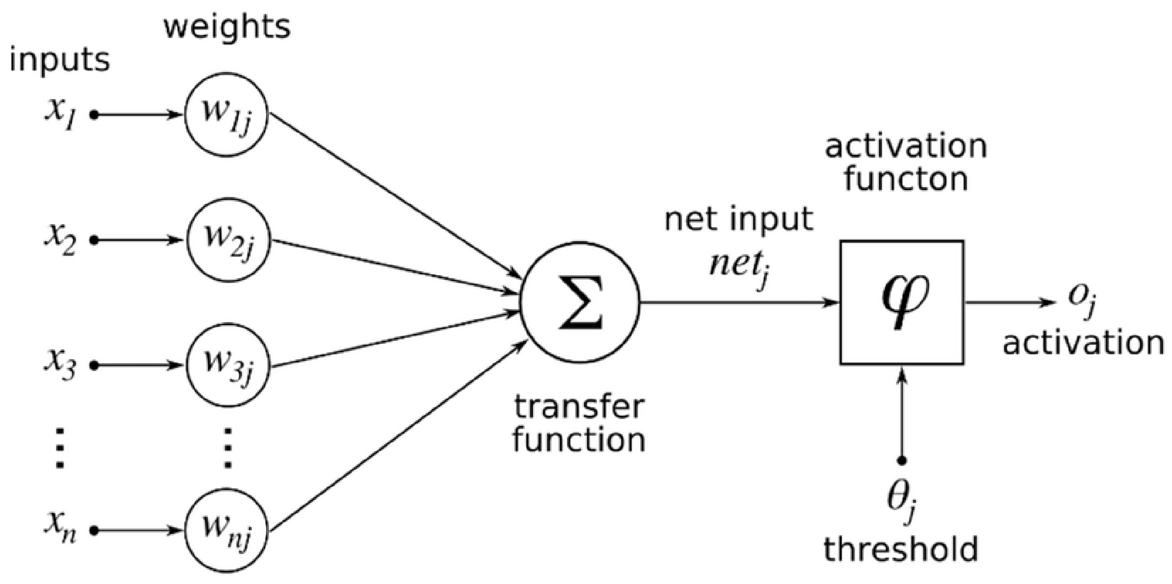


Figure 5 One single neuron with an activation function applied to it

There are several commonly used activation functions such as:

- **Sigmoid activation function**: It maps the input to a value between 0 and 1, it's mostly used in output layers of binary classification problems.
- **ReLU(Rectified Linear Unit) activation function**: It maps all negative values to zero and all positive values to the same positive value, it's mostly used in hidden layers in feedforward neural network
- **Tanh (hyperbolic tangent) activation function**: It maps the input to values between -1 and 1
- **Softmax activation function**: It is commonly used in output layers of multi-class classification problems, it converts the output of the last layer to a probability distribution

Activation functions are applied element-wise, meaning that each element of the input is transformed by the activation function independently.

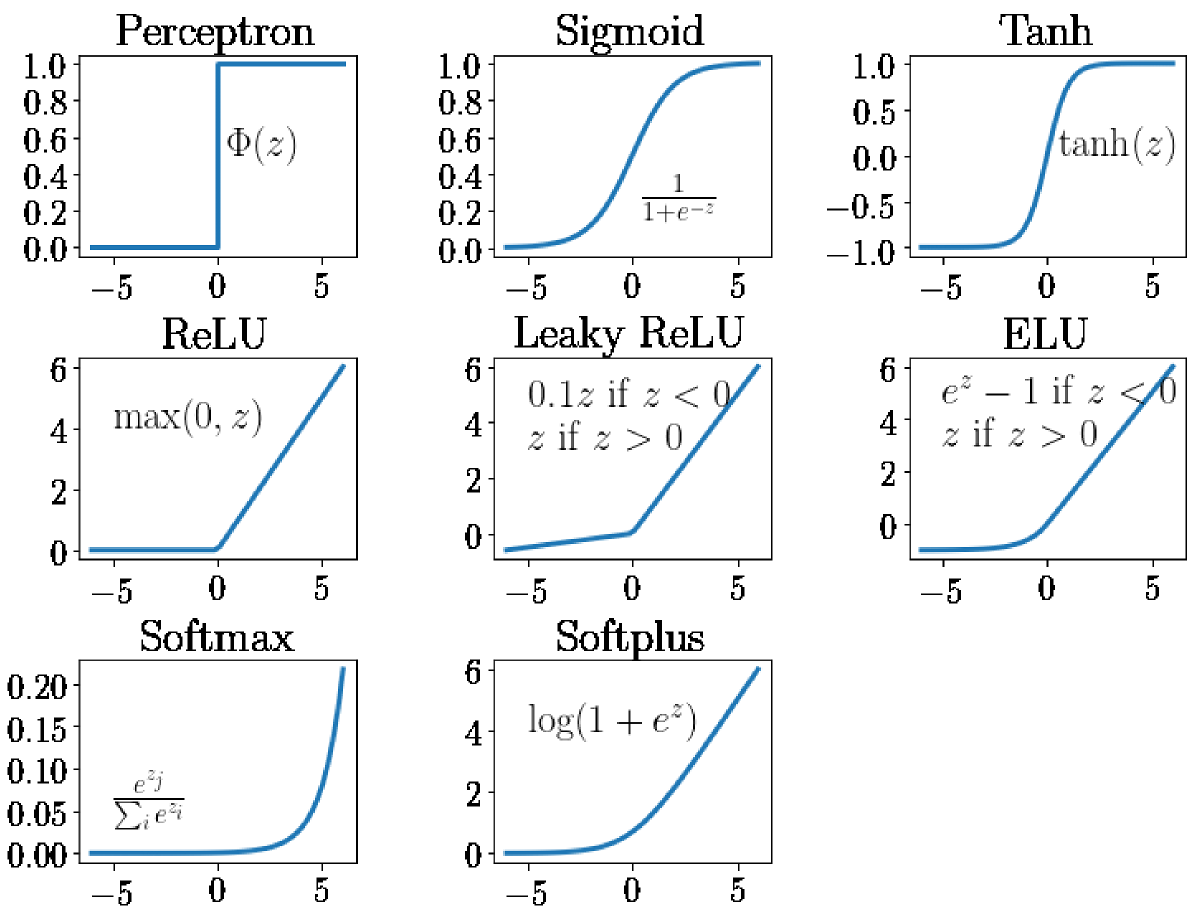


Figure 6 Various activation functions

### 3.1.4 Multilayer perceptron neural network

A multilayer perceptron (MLP) is a type of feedforward artificial neural network that consists of multiple layers of artificial neurons. The input layer receives the input data, and each subsequent layer performs a computation on the input data, using a set of weights and biases that are specific to that layer. The last layer is the output layer, which produces the final output of the network. Between the input and output layers, there may be one or more hidden layers, which perform computations on the input data to extract useful features. The computations performed by the neurons in each layer are usually the same, and are typically a simple mathematical operation, such as a dot product followed by an activation function. The weights and biases of the neurons in each layer are learned during the training process, allowing the network to adjust its computations to improve its performance on the given task.

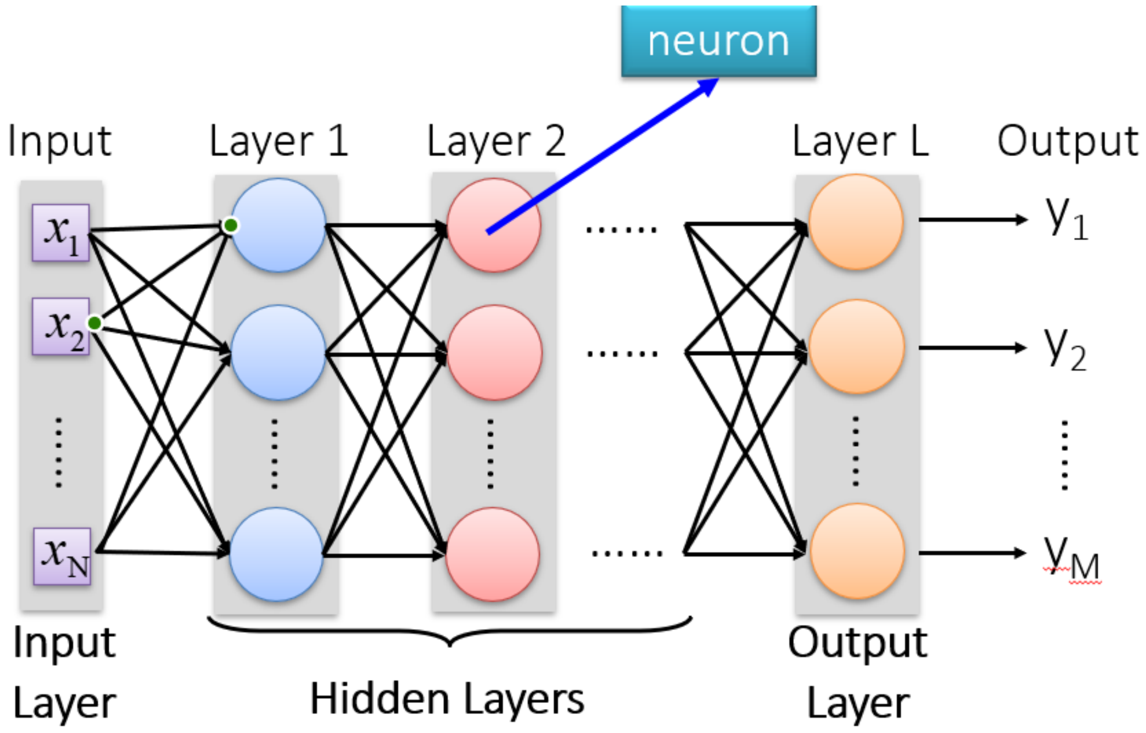


Figure 7 Multilayer perceptron network

### 3.1.5 Loss functions

A loss function in neural networks is a mathematical function that is used to measure the difference between the predicted output of the model and the true output. The goal of training a neural network is to minimize the value of the loss function. Common loss functions include mean squared error, cross-entropy, and hinge loss. The choice of loss function depends on the specific task and the type of data being used.

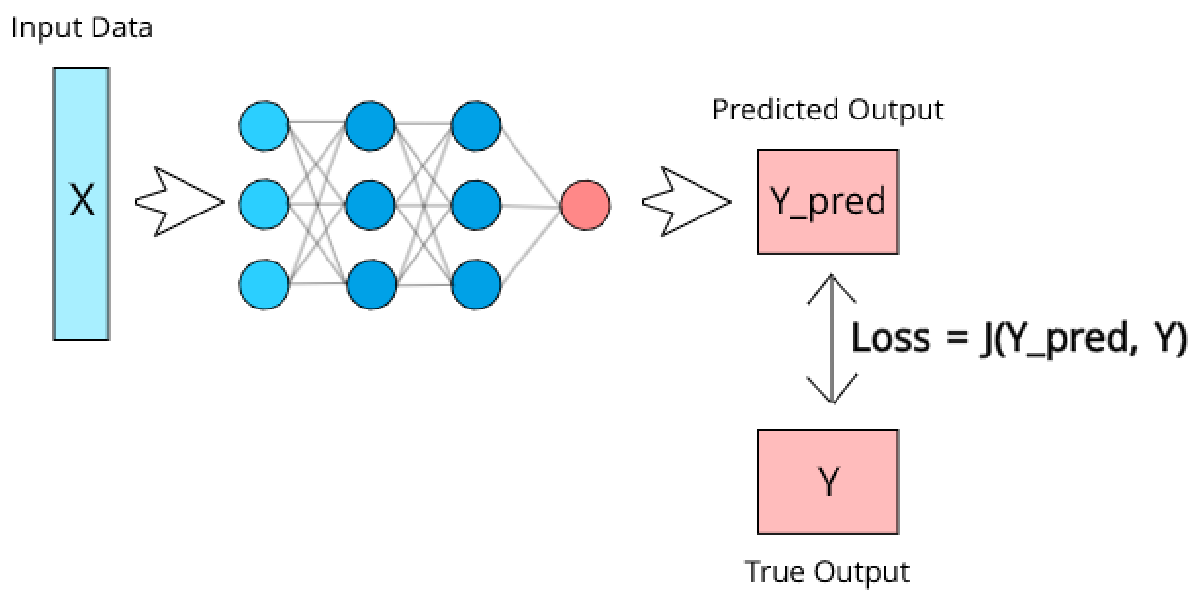


Figure 8 Loss function illustration

### 3.1.6 Optimizers

An optimizer in neural networks is an algorithm used to adjust the parameters of the model in order to minimize the value of the loss function. Optimizers use techniques such as gradient descent to update the model's parameters in a way that reduces the loss. There are several types of optimizers available, each with their own strengths and weaknesses. Some popular optimizers include stochastic gradient descent (SGD), Adam, and RMSprop. The choice of optimizer will depend on the specific task and the characteristics of the data.

### 3.1.7 Training neural networks

Training within neural networks refers to the process of adjusting the parameters of the model in order to improve its performance on a given task. The process of training involves providing the model with a set of inputs and corresponding outputs, called the training data, and using an optimizer to adjust the model's parameters so as to minimize the value of a chosen loss function. The goal of training is to find the set of parameter values that result in the best performance on the task when the model is presented with new, unseen data. This process is done iteratively, adjusting the parameters after each iteration, until the model reaches a satisfactory level of performance.

### 3.1.8 Neural network evaluation

There are several ways to evaluate the performance of a feedforward neural network, some common ones include:

Split the data into training and test sets: A common approach is to use a portion of the available data for training the model, and then evaluate the model's performance on a separate test set.

Metrics such as accuracy, precision, recall, and F1-score: These metrics are commonly used for classification tasks and measure the ability of the model to correctly classify examples.

Cross-validation: This method involves splitting the data into multiple subsets and training the model on different subsets while evaluating on the remaining subsets. This provides a more robust estimate of the model's performance.

Confusion Matrix: This is a table that is used to define the performance of a classification algorithm. It helps in understanding which class the model is predicting correctly and which class it is mis-classifying.

ROC Curve and AUC: Receiver Operating Characteristic (ROC) curve is a graphical representation that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. AUC (Area Under the ROC Curve) represents degree or measure of separability.

Regression Metrics: In case of regression tasks, metrics such as mean squared error, mean absolute error, R-squared, and correlation coefficient can be used to evaluate the performance of the model.

It's important to note that the choice of evaluation method will depend on the specific task and the characteristics of the data.

### 3.1.10 Hyperparameters for feed forward neural network

Hyperparameters in a feedforward neural network refer to parameters that are not learned during training, but are set prior to training. Some common hyperparameters in feedforward neural networks include:

- **Number of layers:** The number of layers in the network, including the input, hidden, and output layers.
- **Number of neurons**: The number of neurons in each layer of the network.

- **Activation function**: The function that is applied to the output of each neuron to introduce non-linearity into the network. Common activation functions include sigmoid, ReLU, and tanh.
- **Learning rate**: The step size at which the optimizer updates the model's parameters.
- **Batch size**: The number of training examples used in each iteration of the training process.
- **Number of epochs**: The number of times the entire training dataset is passed through the network during training.
- **Regularization**: Techniques such as Dropout and L2 regularization to prevent overfitting.
- **Optimizer**: The algorithm used to adjust the model's parameters during training, such as SGD, Adam, or RMSprop.
- **Loss function**: The function used to measure the difference between the predicted output and the true output.

The optimal values for these hyperparameters will depend on the specific task and the characteristics of the data. They can be selected through a process called hyperparameter tuning, which involves training the model with different combinations of hyperparameters and selecting the combination that results in the best performance.

# 4. Project 1) Raining prediction using FNN

### 4.1 Pipeline for feedforward neural networks

The pipeline of a feedforward neural network I utilized involves the following steps:

1) **Data preparation**: This step involves collecting, cleaning, and preprocessing the data to make it suitable for training the model. This can include tasks such as removing missing values, normalizing the data, and converting categorical variables to numerical values.
2) **Model architecture design**: This step involves choosing the number of layers, the number of neurons in each layer, and the type of activation function to use.
3) **Model training**: This step involves using an optimizer to adjust the model's

parameters to minimize the value of the chosen loss function using the training data.

4) **Model evaluation**: This step involves using one or more evaluation metrics to assess the performance of the trained model on a separate test set or by using cross-validation techniques.
5) **Hyperparameter tuning**: This step involves adjusting the hyperparameters of the model such as learning rate, batch size and number of epochs to improve the model's performance.
6) **Model deployment**: Once the model has been trained and evaluated, it can be deployed to perform predictions on new, unseen data.
7) **Model monitoring**: After the model is deployed, it is important to monitor its performance over time and retrain the model if necessary.

It's important to note that the pipeline may vary depending on the specific task and the characteristics of the data, but the general process of data preparation, model design, training, evaluation and deployment remains the same.

## 4.2 Project description

In this minor project I decided to with the of use of one the simplest network models: the multilayer perceptron (MLP) feed forward neural network, to build and apply it to a historic **Australian weather** dataset to build and train my own simple feed forward network which will give prediction whatever it will rain the next day or not depending on the air humidity.  Pipeline described in section 4.1 will be applied. The goal with this project is to illustrate that it is possible to accomplish remarkable results with a very simple architecture.

### 4.2.1 Data description

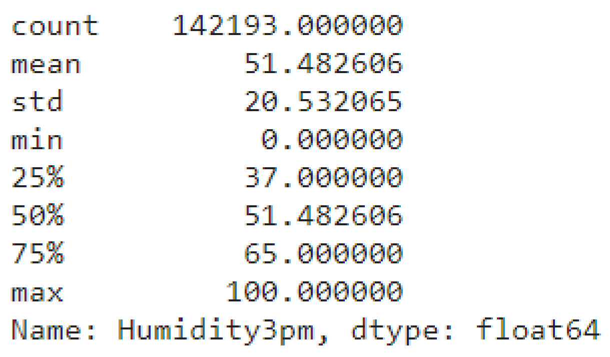
The following dataset is used in the project: the weatherAUS dataset. It is a comprehensive collection of daily weather observations from various locations across Australia. It includes information such as temperature, precipitation, wind direction and speed, and other meteorological measurements.

The weaterAUS dataset is an enormous data frame consisting of over 145,000 daily observations from over 45 Australian weather stations. Since the goal of the given assignment was mainly to focus on learning and not achieving as good results as possible, simplifications were made by only taking into account the **Humidity3pm** feature for trying to predict the **Raintomorrow** variable**,** which is the target variable in my case.

- The **Humidity3pm** coefficient describes the humidity in percentage at random days at 3pm
- The **RainTomorrow** parameter describes whether it rained or not the following day

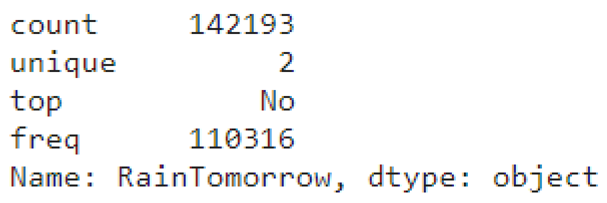
### 4.2.2 Data preparation

Even though the data were collected and stored into a CSV file some data cleaning/adjustments had to be made to the data frame. Primary I removed all the cloumns with NAN values:



```
count    142193.000000
mean         51.482606
std          20.532065
min           0.000000
25%          37.000000
50%          51.482606
75%          65.000000
max         100.000000
Name: Humidity3pm, dtype: float64
```

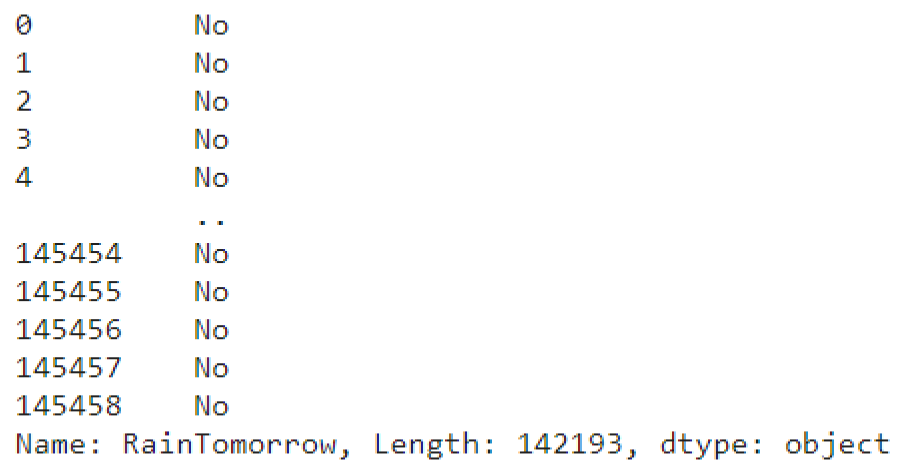
Figure 9 statistical summary of Humidity3pm feature



```
count     142193
unique         2
top           No
freq      110316
Name: RainTomorrow, dtype: object
```

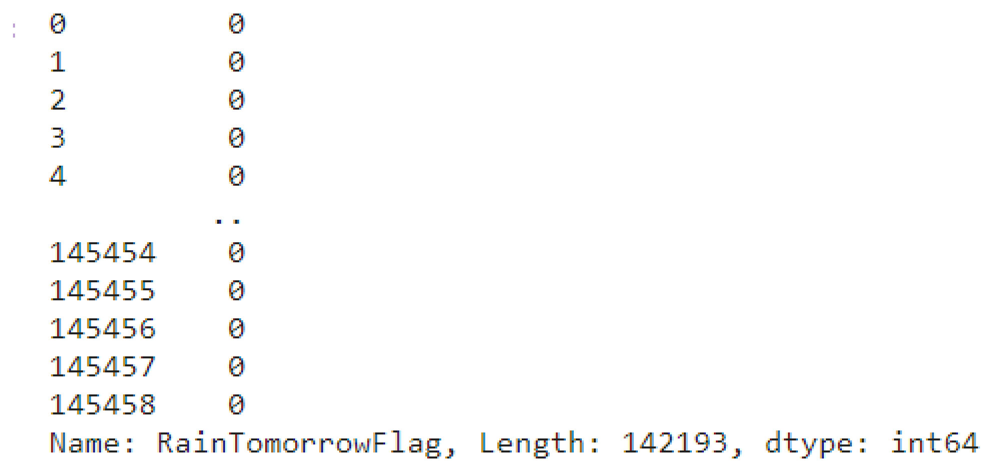
Figure 10 statistical summary of RainTomorrow label

Also in order to be able process the **Raintomorrow** data into the model a flag for making the data into int datatype was made. Where a 0 value indicates that it did not rain the next day (the no value), and a 1 for the yes value which says that it will probably rain then next day:



```
0         No
1         No
2         No
3         No
4         No
          ..
145454    No
145455    No
145456    No
145457    No
145458    No
Name: RainTomorrow, Length: 142193, dtype: object
```

Figure 11 Raintomorrow data before flag processing of data



```
0         0
1         0
2         0
3         0
4         0
         ..
145454    0
145455    0
145456    0
145457    0
145458    0
Name: RainTomorrowFlag, Length: 142193, dtype: int64
```

Figure 12 Raintomorrow data after flag processing of data

### 4.2.3 Model architecture design

The chosen model's architecture is as simple as possible. consisting of one input node in the input layer, just two hidden nodes at the dense layer (a Softplus activation function is also applied after the output from the hidden layer), and one neuron at the output layer (where a sigmoid activation function is used to give the probability value of the classification) to classify whether it will rain or not.

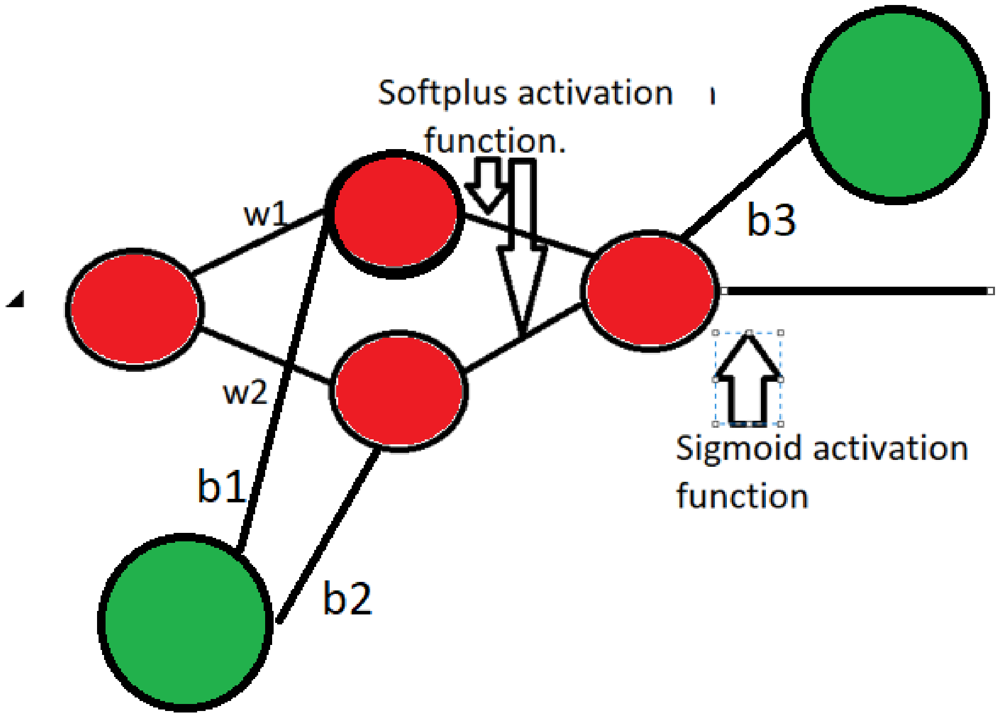


Figure 13 Used feed forward network model structure for project 1

The total number of trainable parameters in the network may be calculated by multiplying the number of connections between each layer by an additional one for each bias for each corresponding neuron. At input layer to hidden layer theres 1 x 2 + 2 (w1+w2+b1+b2) = 4 parameters, and from the hidden layer to output layer there's 2 x 1 + 1 = 3 parameters . The total number of trainable parameters will therefore be 3 + 4 = 7. Also shown in the model structure summary below:

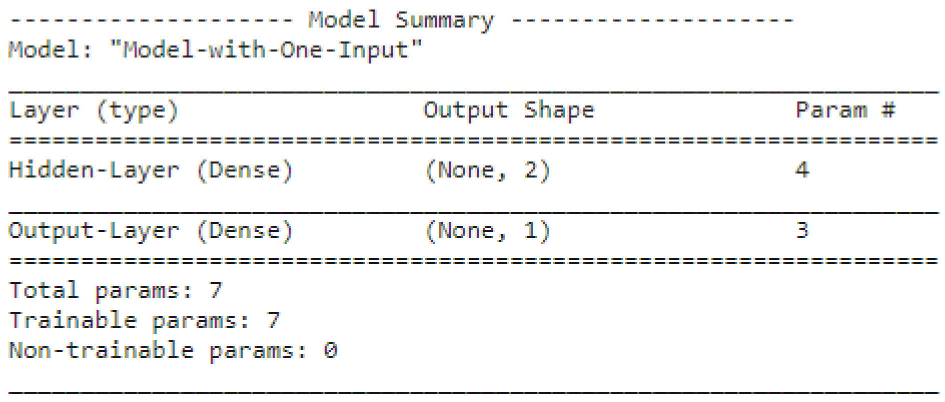

```
-------------------- Model Summary --------------------
Model: "Model-with-One-Input"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Hidden-Layer (Dense)         (None, 2)                 4
_________________________________________________________________
Output-Layer (Dense)         (None, 1)                 3
=================================================================
Total params: 7
Trainable params: 7
Non-trainable params: 0
_________________________________________________________________
```

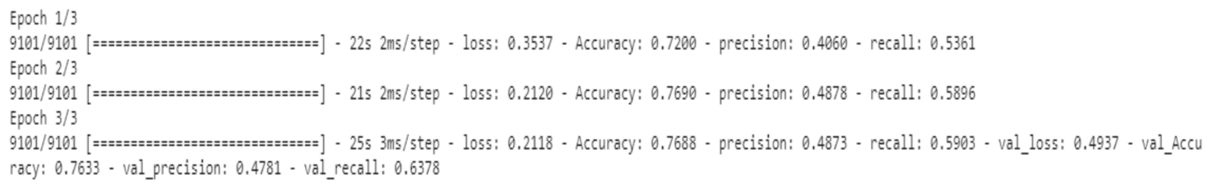
Figure 14 total number of parameters , model summary

### 4.2.4 Model configuration:

Before fitting the data into the model the the "compile" step in Keras is used to configure the learning process of a model by specifying the optimizer, loss function, and metrics. This information is used during the training process to update the model's weights and evaluate its performance. The chosen optimizer was the Adam optimizer, which is a gradient descent optimization algorithm.  It is an extension of the basic stochastic gradient descent algorithm. The chosen loss function was the binary cross entropy. This loss function is suitable because the binary classification problem. The metrics used for evaluating the model was Accuracy, Precision and recall.

### 4.2.5 Model training

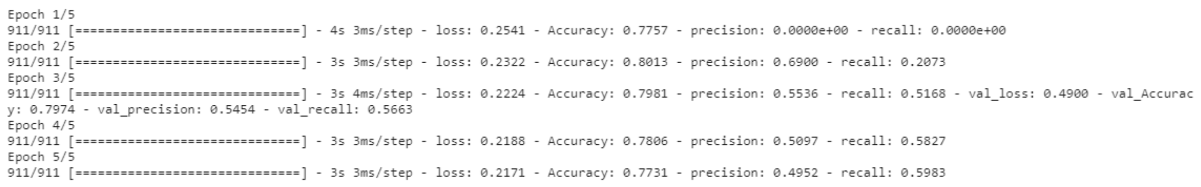
Before fitting and training the model on the Australian weather dataset a few hyperparameters were selected. The number of samples per gradient update or batch size was set to 10. The total number of complete passes through the entire training dataset or epochs were set to 3. During each epoch, the model's parameters are updated in order to minimize the error on the training set.



```
Epoch 1/3
9101/9101 [==============================] - 22s 2ms/step - loss: 0.3537 - Accuracy: 0.7200 - precision: 0.4060 - recall: 0.5361
Epoch 2/3
9101/9101 [==============================] - 21s 2ms/step - loss: 0.2120 - Accuracy: 0.7690 - precision: 0.4878 - recall: 0.5896
Epoch 3/3
9101/9101 [==============================] - 25s 3ms/step - loss: 0.2118 - Accuracy: 0.7688 - precision: 0.4873 - recall: 0.5903 - val_loss: 0.4937 - val_Accu
racy: 0.7633 - val_precision: 0.4781 - val_recall: 0.6378
```

**Figure 15 Training process with epoch value of 3 and batch size of 10**

After the first training attempt another training session were made, to wish for further accuracy improvements. By changing the number of epochs (switching it to 5) and batch ( to a batch size of 100) sizes a small increase in accuracy was given:

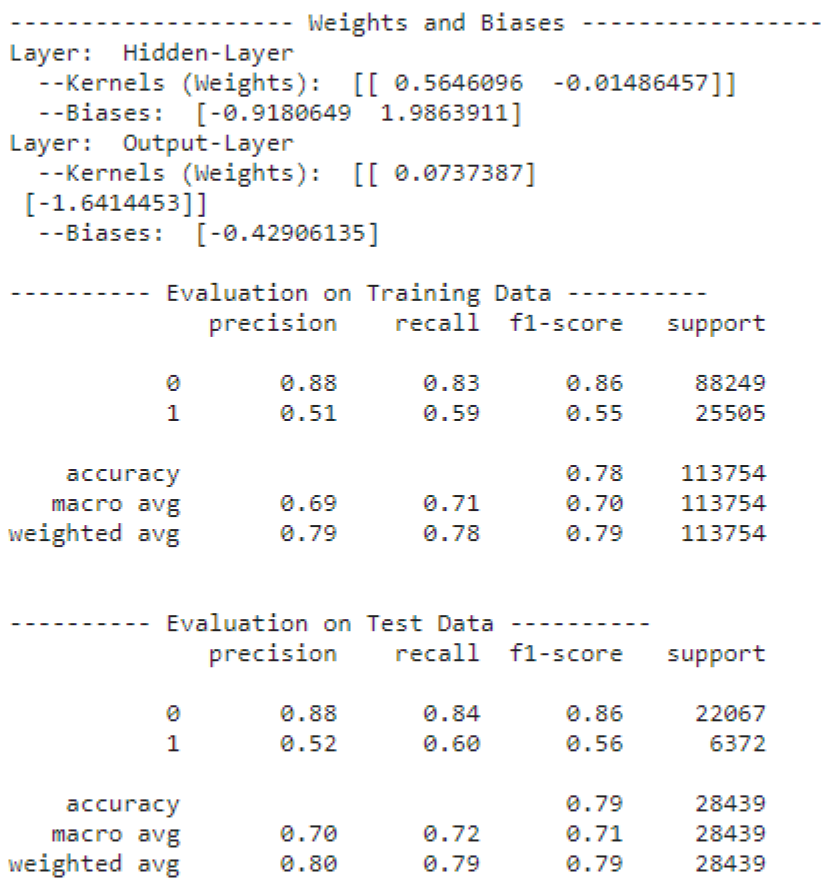


```
Epoch 1/5
911/911 [==============================] - 4s 3ms/step - loss: 0.2541 - Accuracy: 0.7757 - precision: 0.0000e+00 - recall: 0.0000e+00
Epoch 2/5
911/911 [==============================] - 3s 3ms/step - loss: 0.2322 - Accuracy: 0.8013 - precision: 0.6900 - recall: 0.2073
Epoch 3/5
911/911 [==============================] - 3s 4ms/step - loss: 0.2224 - Accuracy: 0.7981 - precision: 0.5536 - recall: 0.5168 - val_loss: 0.4900 - val_Accurac
y: 0.7974 - val_precision: 0.5454 - val_recall: 0.5663
Epoch 4/5
911/911 [==============================] - 3s 3ms/step - loss: 0.2188 - Accuracy: 0.7806 - precision: 0.5097 - recall: 0.5827
Epoch 5/5
911/911 [==============================] - 3s 3ms/step - loss: 0.2171 - Accuracy: 0.7731 - precision: 0.4952 - recall: 0.5983
```

**Figure 16 Training process with epoch value of 5 and batch size of 100**

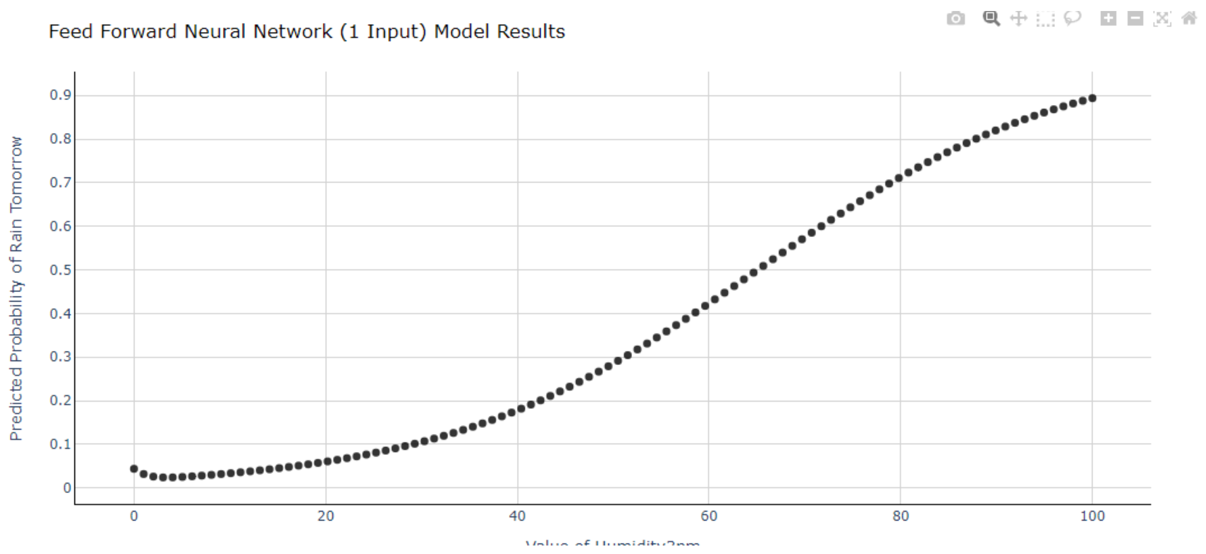
### 4.2.6 Model evaluation

The trained weights and biases parameter values in the final model with its given metrics result:



```
-------------------- Weights and Biases -----------------
Layer:  Hidden-Layer
  --Kernels (Weights):  [[ 0.5646096  -0.01486457]]
  --Biases:  [-0.9180649  1.9863911]
Layer:  Output-Layer
  --Kernels (Weights):  [[ 0.0737387]
 [-1.6414453]]
  --Biases:  [-0.42906135]

---------- Evaluation on Training Data ----------
              precision    recall  f1-score   support

           0       0.88      0.83      0.86     88249
           1       0.51      0.59      0.55     25505

    accuracy                           0.78    113754
   macro avg       0.69      0.71      0.70    113754
weighted avg       0.79      0.78      0.79    113754


---------- Evaluation on Test Data ----------
              precision    recall  f1-score   support

           0       0.88      0.84      0.86     22067
           1       0.52      0.60      0.56      6372

    accuracy                           0.79     28439
   macro avg       0.70      0.72      0.71     28439
weighted avg       0.80      0.79      0.79     28439
```

Figure 17 Weigth , biases and evluation matrix



## Results

Even though the simplest possible sequential model was used with only 1 hidden dense layer with 2 nodes. The network still managed to result in very slightly improvements after training. As almost all the evaluation of the test set results were a bit higher than the pretrained model. However, the model managed to correctly end up with a precision of 88 % on the test-set with prediction of that if the humidity at 3 is at certain value that likelihood of *not* raining (0 value as target variable)would be the righ prediction for the next day with only use of one humidity feature. The precision of predicting right label value

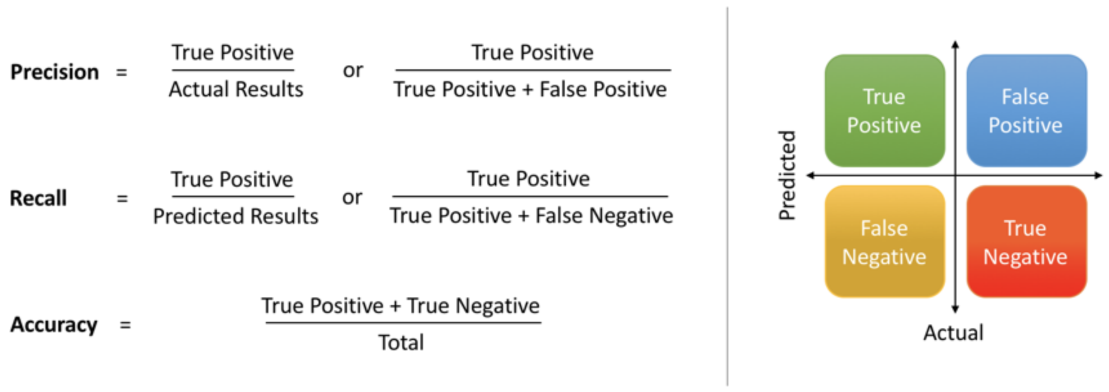

Precision = True Positive / Actual Results or True Positive / True Positive + False Positive
Recall = True Positive / Predicted Results or True Positive / True Positive + False Negative
Accuracy = True Positive + True Negative / Total
Predicted
True Positive
False Positive
False Negative
True Negative
Actual

**Figure 18 Precision, recall and accuracy calculations**

# 5. Convolution neural network theory

A convolutional neural network (**CNN**) is a type of neural network architecture specifically designed to process and analyze images. A standard CNN network is commonly split up into two parts: **Feature extraction** section refers to the process of extracting features or characteristics from the input image using convolutional layers. These convolutional layers apply filters to the image, which are then used to detect features such as edges, shapes, and textures. The output of these layers is a set of feature maps, which are then passed on to the next set of layers in the network for further processing and classification. The other part of the CNN model is the **classification** part which typically is a fully connected layer, also known as a dense layer, that is trained to map the output of the convolutional and pooling layers to a set of class scores. These class scores represent the likelihood that a given input image belongs to each class. The class with the highest score is then chosen as the final classification for that image. A CNN   It is composed of multiple layers, including :

- **Convolutional Layers**
- **Pooling Layers**
- **Normalization Layers**
- **Fully Connected Layers**

- **Activation functions**

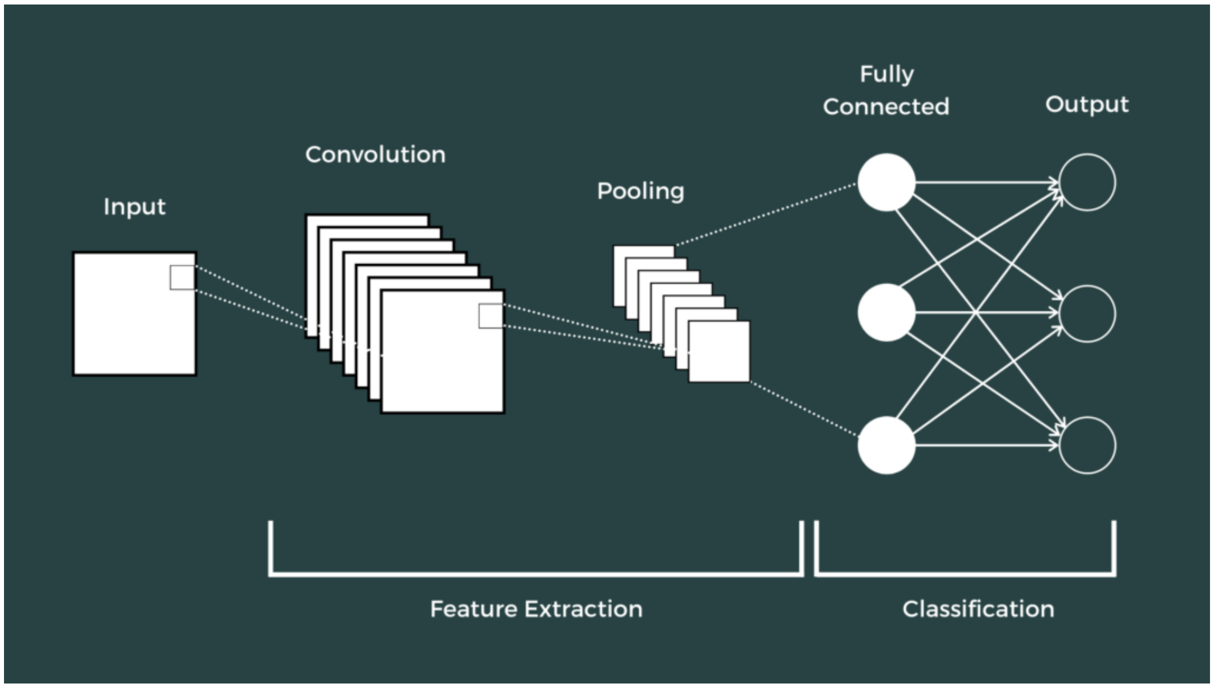


Figure 19 Convolutional neural network

### 5.2 Convolution layer

The purpose of convolutional layers in a CNN is to extract relevant features from the input image. Convolutional layers use a set of learnable filters, which are small matrices of weights, to scan the input image in a sliding window fashion. The filters are designed to detect specific features in the image, such as edges, corners, and textures

By using multiple filters in multiple convolutional layers, a CNN can extract a wide variety of features from the input image, including both low-level features such as edges and high-level features such as objects. These features are then passed through the other layers of the network, such as pooling layers and fully connected layers, to make a final prediction or classification.

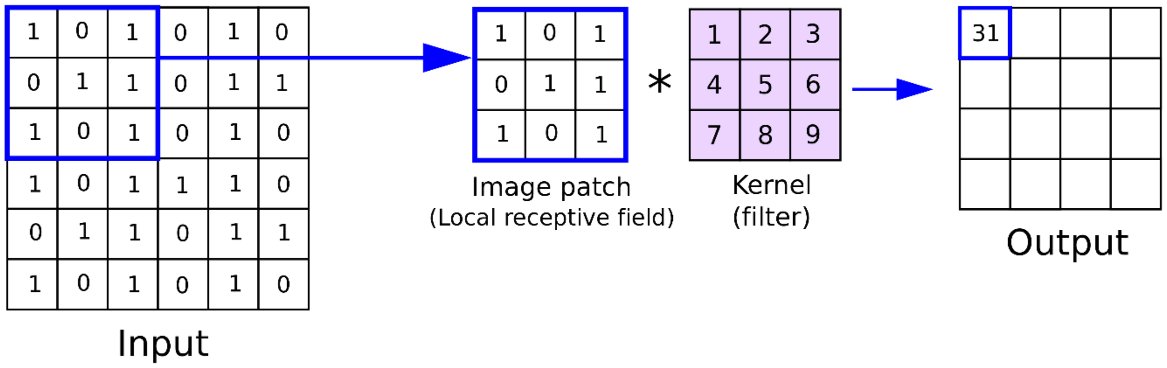


Figure 20 Convolutional operation with a 3x3 filter

The figure above illustrates a convolutional operation on a 6 (height) x 6 (width) x 1 (number of channels) resolution input image with a 3 x 3 x 1 kernel or filter applied to it**.** As the filters traverse over the whole image, a mathematical operation called convolution is performed, which combines the values of the pixels in the image with the values of the weights in the filter. The output of this operation is a new matrix called a feature map, which represents the presence of the specific feature detected by the filter. A hyperparameter named stride is set in the beginning. It determines the step size of the convolution operation and decides the spatial resolution of the feature map(s). The default value for stride is normally 1, which indicates that the filter is sliding one pixel into a new image patch. Then the filter is repeatedly sliding and computing the convolution operation again until whole image has been covered. In case if the image. As the main goal of the convolution operation is to extract high-level features such as edges, color, and gradient orientation, preferably multiple convolution layers are utilized. Another hyperparameter is the padding variable. This hyperparameter will augment or pad the input image to either increase the resolution or maintain the resolution of the feature maps at the same level as the input size. This is called "**same padding."** There's also other padding called **valid padding**, which implies no padding at all and the output size will therefore change.

$$\text{output size} = \left\lfloor \frac{(\text{input size}) + 2*\text{padding} - (\text{kernel size} - 1) - 1}{\text{stride}} + 1 \right\rfloor$$

Figure 21 output size formula with valid padding

## 5.3 Pooling layers

These layers are typically placed after the convolutional layers and serve to reduce the spatial dimensions of the feature maps. This is done by applying a down-sampling operation to the feature maps, such as max pooling, which selects the maximum value from a small window of the feature map. This reduces the size of the feature maps, while also reducing the number of parameters in the model. Pooling layers also help in making the model more robust to small translations and deformations in the input image.

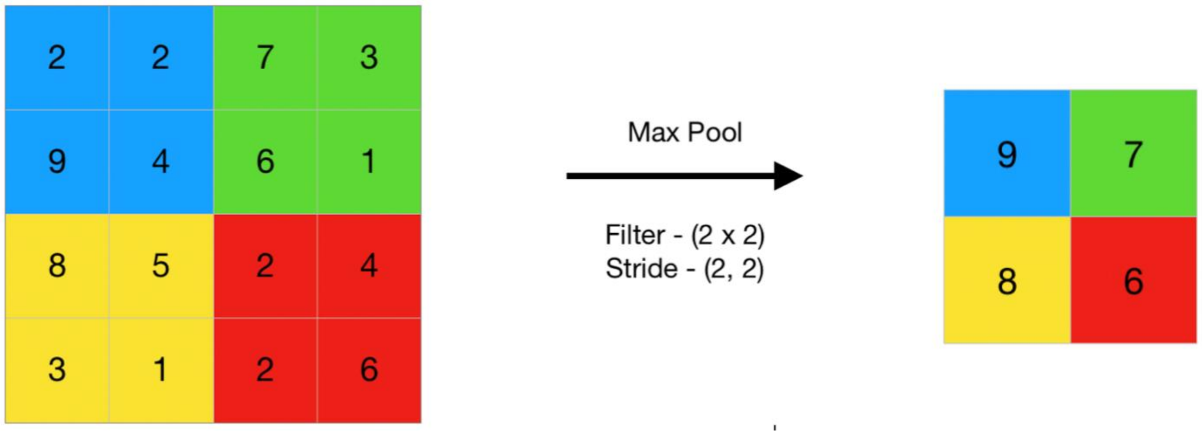


Figure 22 Max pooling operation with a 2x2 filter and 2 stride

## 5.4 Normalization Layers

These layers are used to normalize the output of the convolutional layers and are typically placed after the convolutional layers. They are used to speed up the training process, improve the stability of the model, and prevent overfitting. Commonly used normalization techniques are Batch Normalization, Layer Normalization etc.

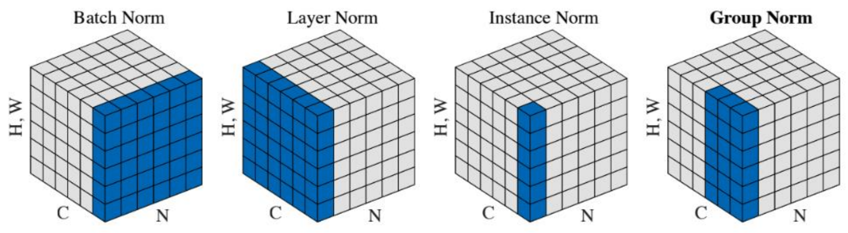


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with $N$ as the batch axis, $C$ as the channel axis, and $(H, W)$ as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

Figure 23 Different kind of normalization methods

## 5.5 Fully connected layers

These layers are placed at the end of the CNN and are used to make a final prediction or classification. The layers are called fully connected because they are connected to all the neurons in the previous layer, just like a normal multilayer perceptron network. These layers take the output from the pooling layers and use it to make a prediction by using techniques like backpropagation. The output of the final fully connected layer is usually passed through a softmax activation function, which produces a probability distribution over the possible classes.

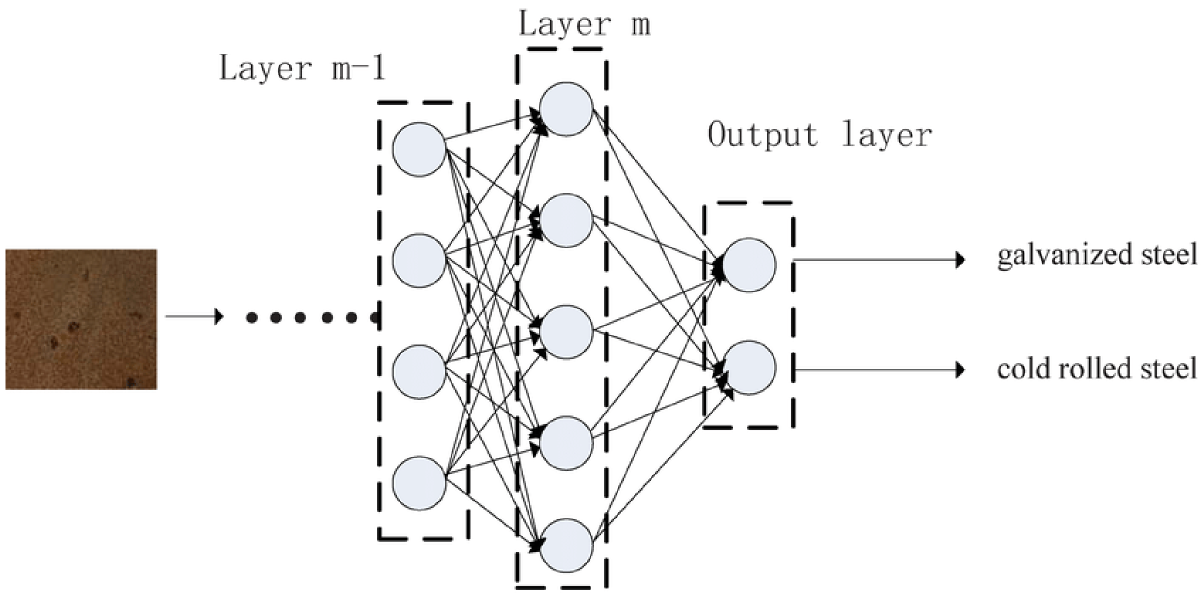


Figure 24 Fully connected layer in CNN

## 5.6 Activation funcctions

These functions are applied element-wise to the output of the convolutional layer and are used to introduce non-linearity into the model. Commonly used activation functions are ReLU, sigmoid, tanh etc. ReLU (Rectified Linear Unit) is a popular choice as it helps to improve the training speed of the model by introducing non-linearity and also prevents the vanishing gradient problem

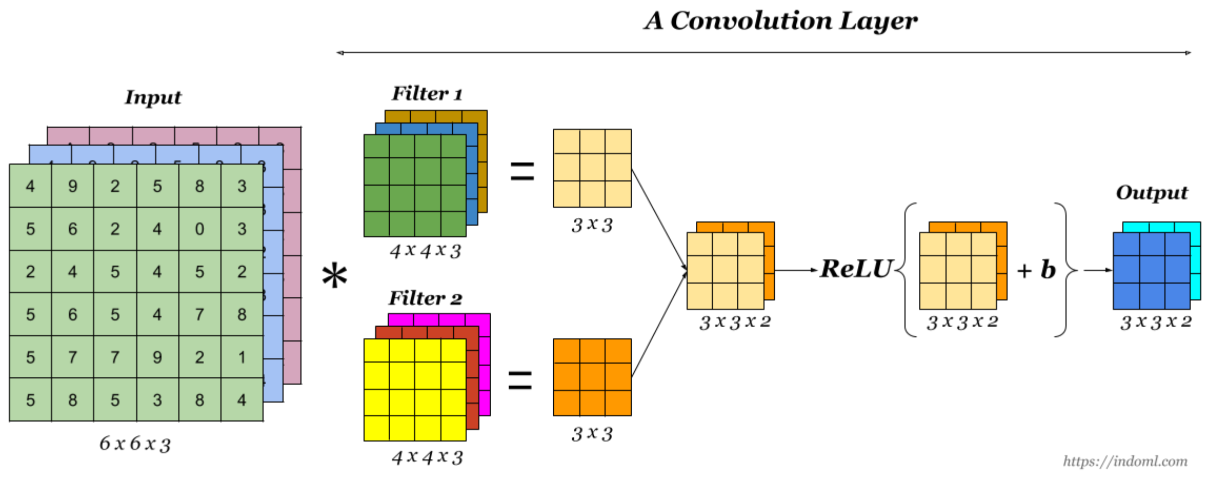


Figure 25 ReLU activation function

**Data augmentation** is a technique used to artificially increase the size of a dataset by applying a set of predefined transformations to the existing data. This can help to prevent overfitting in neural networks by providing the model with more diverse examples to learn from. Common data augmentation techniques include flipping, rotating, cropping, and zooming images, as well as adding noise to audio or text data.

**Dropout**

Dropout is a regularization technique used in neural networks to reduce overfitting. The basic idea behind dropout is to randomly drop out, or set to zero, some of the neurons in the network during training. This is done by applying a dropout mask, which is a binary mask, to the activations of the neurons. The mask is created by randomly setting a certain proportion of the activations to zero. This proportion is known as the dropout rate and is typically set between 0.2 and 0.5. When a neuron is dropped out, it is removed from the network for the current training iteration. This means that the other neurons in the network will have to compensate for the dropped out neuron's contribution to the final output. This has the effect of forcing the network to learn more robust and less specialized features. Dropout is typically applied only during training, not during testing or inference, since it is only necessary to prevent overfitting during training. Dropout is usually applied on fully connected layers and convolutional layers, but not on input or output layers. Dropout can improve the performance of a neural network by reducing overfitting, but it can also make the model less accurate if used excessively or in the wrong places. Therefore, it is important to use it properly and find the right dropout rate for your specific problem.

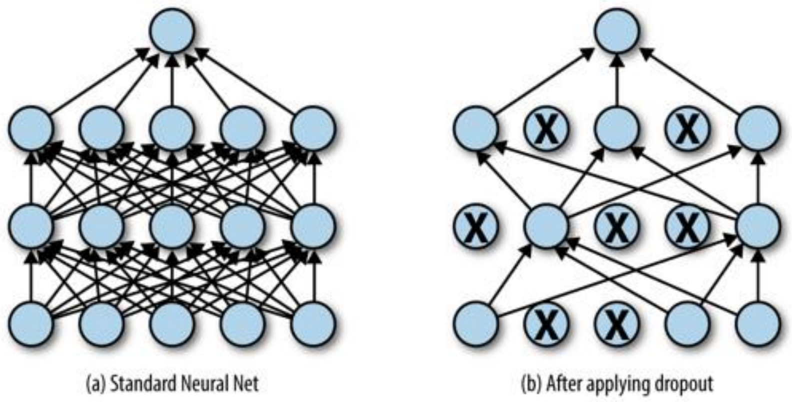


Figure 26 Dropout regularization technique

## Transfer learning and fine tuning

Transfer learning is a technique in which a pre-trained model, typically trained on a large dataset, is used as the starting point to train a new model on a different but related task. The idea is to leverage the knowledge learned by the pre-trained model, which can save computational resources and improve the performance of the new model.
There are two main types of transfer learning: feature-based and fine-tuning.
In feature-based transfer learning, the pre-trained model's layers, which extract features from the input data, are used as a fixed feature extractor. The output of these layers is fed as input to a new model, typically a classifier, which is trained to perform the target task. This approach is useful when the input data of the target task is similar to the data the pre-trained model was trained on, but the output data is different.

In fine-tuning transfer learning, the pre-trained model is further trained on the target task. This is done by unfreezing some or all of the layers of the pre-trained model and training them on the new data along with the new layers added on top. This approach is useful when the target task and the pre-trained model's task are similar and there is a limited amount of labeled data available for the target task.

Transfer learning has been used to achieve state-of-the-art results in a wide range of tasks including image classification, object detection, and natural language processing. Some popular pre-trained models that can be used for transfer learning include VGG, ResNet, and BERT.

# 6. Project 2) Image classification on Fashion-MNIST

**6.1 Project description**

In this minor project I decided to with the of use a modification of one the most famous and successful convolution neural network model architecture: the VGG 16 CNN. With the goal of to build and apply dropout regularization technique and some data augmentation techniques to compare the results on the world famous Fashion-MNIST dataset to train and in the end classify different cloths with a test set and discuss the results.

**6.2 Data description**

The following dataset used in the project is the Fashion-MNIST dataset. This dataset contains images of clothing items, such as shirts, pants, and shoes, and is meant to be a more challenging dataset for machine learning models to classify. The dataset contains 60,000 training images and 10,000 test images, with each image having a resolution of 28x28 pixels. Each image is labeled with one of 10 classes, which correspond to different types of clothing items: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot. The pixel values of each image are normalized and the data is divided in train and test dataset with 60000 and 10000 samples respectively.

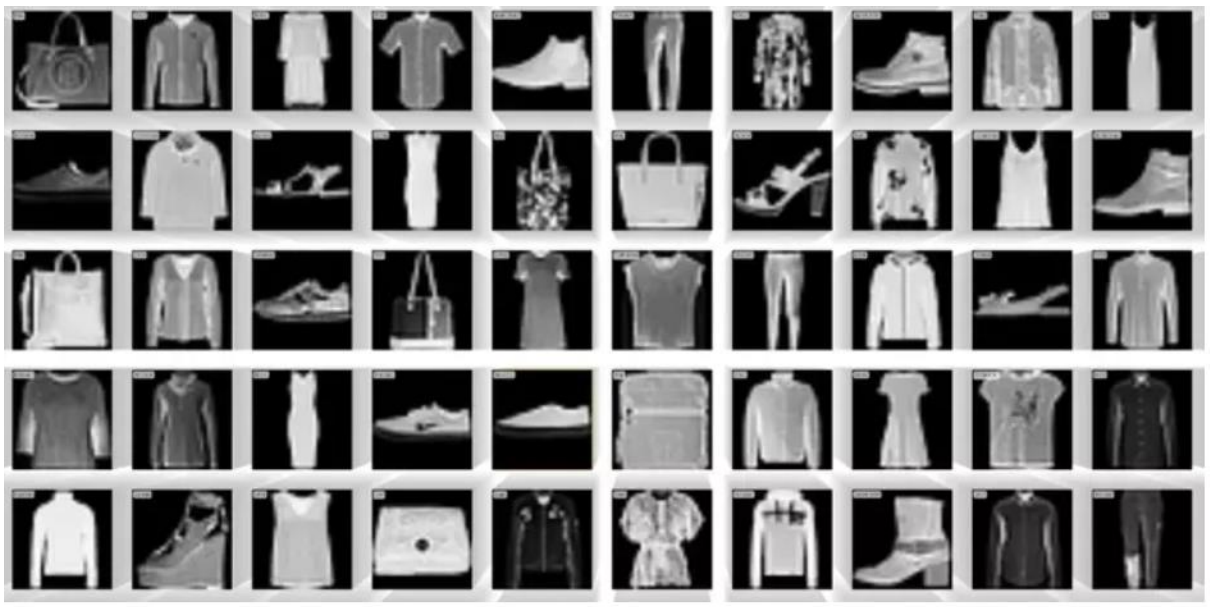


Figure 27 , 40 samples of fashion-MNIST dataset

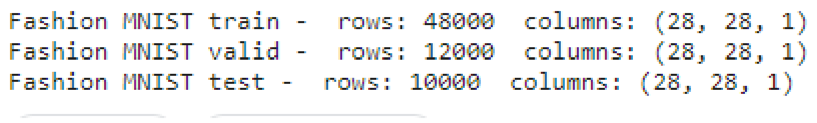
### 6.3 Image description

Each image consists of an image with 28 (width) x 28 (height)= 784 grayscale pixels in total, where each pixel value describes the illumination level within the range of 0 to 255 (0 indicates total darkness while 255 means full lightness ). However, the training and test set obth have 785 columns. As this is a supervised classification problem the first column within each image is indicating the labeled or class the image belongs to. This is what the desired prediction label we want to predict from the test-set and classify each garment into right class.

### 6.4 Data preprocessing

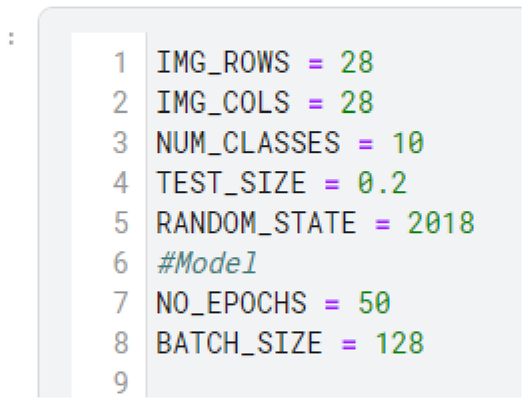
Initially a column reshape had to be made from a 784 column into a 28x28x1 (Width * Height * number of channels ,1 in this case because of greyscale). Also a separation of the target label as a unique vector.´ The dataset were also split into three different subsets:

- Test-set with 10000 unique images .
- Validation-set splits up the training set of the 60000 images. In this case a split covering of 20 % were performed of the train set equivalent to 60000 * 0,2 = 12000 images
- Train-set consists of the remaining set of the trainset – Validation set = 60000-12000 = 48000 different images to train the network on.

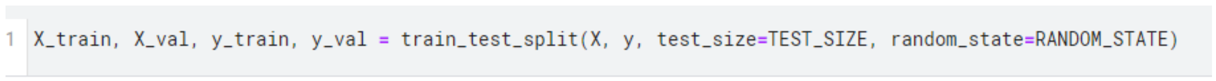


```
Fashion MNIST train -  rows: 48000  columns: (28, 28, 1)
Fashion MNIST valid -  rows: 12000  columns: (28, 28, 1)
Fashion MNIST test -  rows: 10000  columns: (28, 28, 1)
```

## Parameters



```
IMG_ROWS = 28
IMG_COLS = 28
NUM_CLASSES = 10
TEST_SIZE = 0.2
RANDOM_STATE = 2018
#Model
NO_EPOCHS = 50
BATCH_SIZE = 128

```



```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=TEST_SIZE, random_state=RANDOM_STATE)
```

### 6.5 Model architecture design without dropout

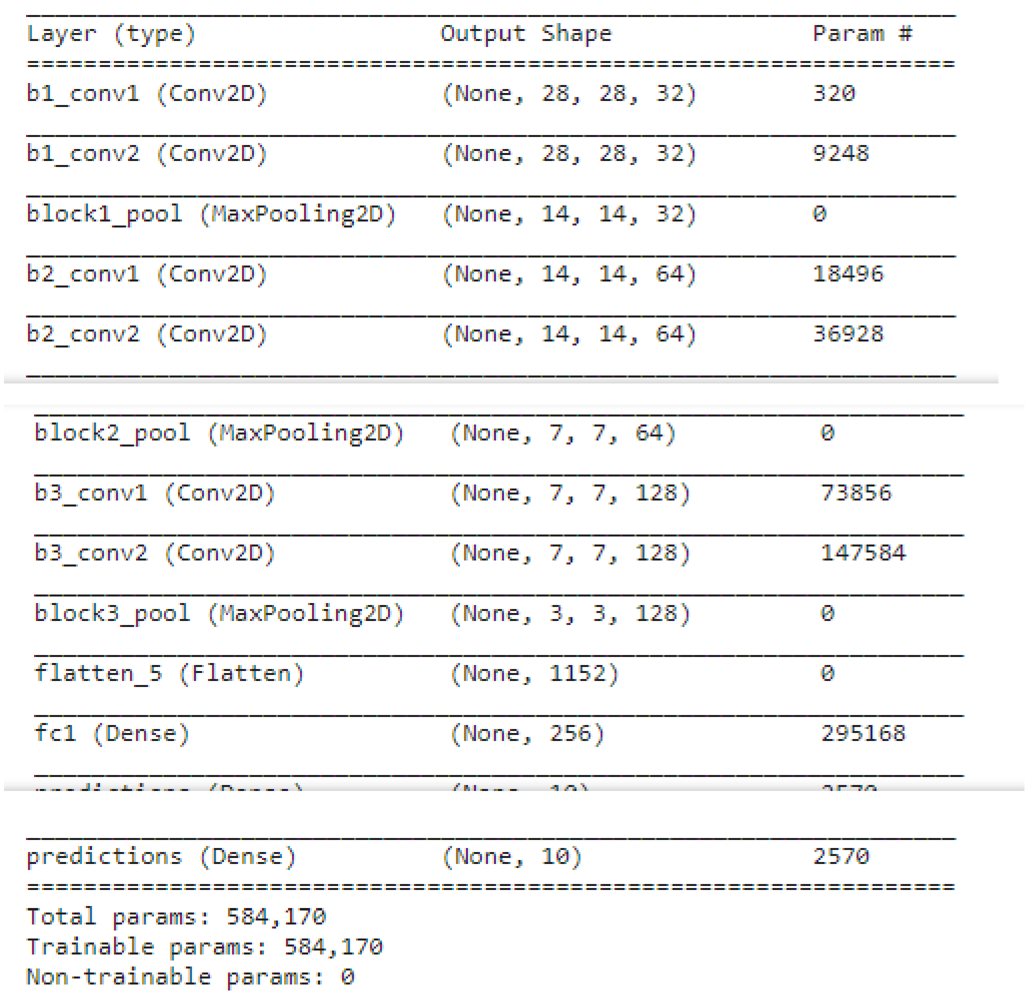
The chosen model architecture used in this project is inspired by the **VGG16** architecture, suitable for the fashion-MNIST dataset. The VGG16 CNN architecture is composed of a series of convolutional and max pooling layers, followed by a few fully connected layers. The architecture used can be divided into two main parts: the convolutional part and the fully connected part.

In the convolutional part some modifications and simplifications were made of the VGG16 instead of being made up of 13 convolutional layers 6 convolutions layer are used, with the following structure:

- The first two layers are convolutional layers with 32 filters of size 3x3, followed by a ReLU activation function and a max pooling layer with a 2x2 filter and stride 2.
- The next two layers are composed of two convolutional layers with 64 filters of size 3x3, followed by a ReLU activation function and a max pooling layer with a 2x2 filter and stride 2.
- The last two convolution layers are composed with 128 filters of size 3x3, followed by a ReLU activation function and a max pooling layer with a 2x2 filter and stride 2.

The fully connected part of my model is made up of three fully connected layers, with the following structure:

- The first fully connected layer has 256 neurons and a ReLU activation function.
- The last fully connected layer has 10 neurons and a softmax activation function, used for the classification task.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
b1_conv1 (Conv2D)            (None, 28, 28, 32)        320       
_________________________________________________________________
b1_conv2 (Conv2D)            (None, 28, 28, 32)        9248      
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 14, 14, 32)        0         
_________________________________________________________________
b2_conv1 (Conv2D)            (None, 14, 14, 64)        18496     
_________________________________________________________________
b2_conv2 (Conv2D)            (None, 14, 14, 64)        36928     
_________________________________________________________________
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 7, 7, 64)          0         
_________________________________________________________________
b3_conv1 (Conv2D)            (None, 7, 7, 128)         73856     
_________________________________________________________________
b3_conv2 (Conv2D)            (None, 7, 7, 128)         147584    
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 3, 3, 128)         0         
_________________________________________________________________
flatten_5 (Flatten)          (None, 1152)              0         
_________________________________________________________________
fc1 (Dense)                  (None, 256)               295168    
_________________________________________________________________
_________________________________________________________________
predictions (Dense)          (None, 10)                2570      
=================================================================
Total params: 584,170
Trainable params: 584,170
Non-trainable params: 0
```

Figure 28 CNN model structure used in project 2

The total number of parameters in the network is 584 170. This is a rather small number for today's deep-neural networks. However, the total number of parameters are depending on many different parameters such as Kernel size, stride size, number of filters, padding size, pooling size, and number of layers.

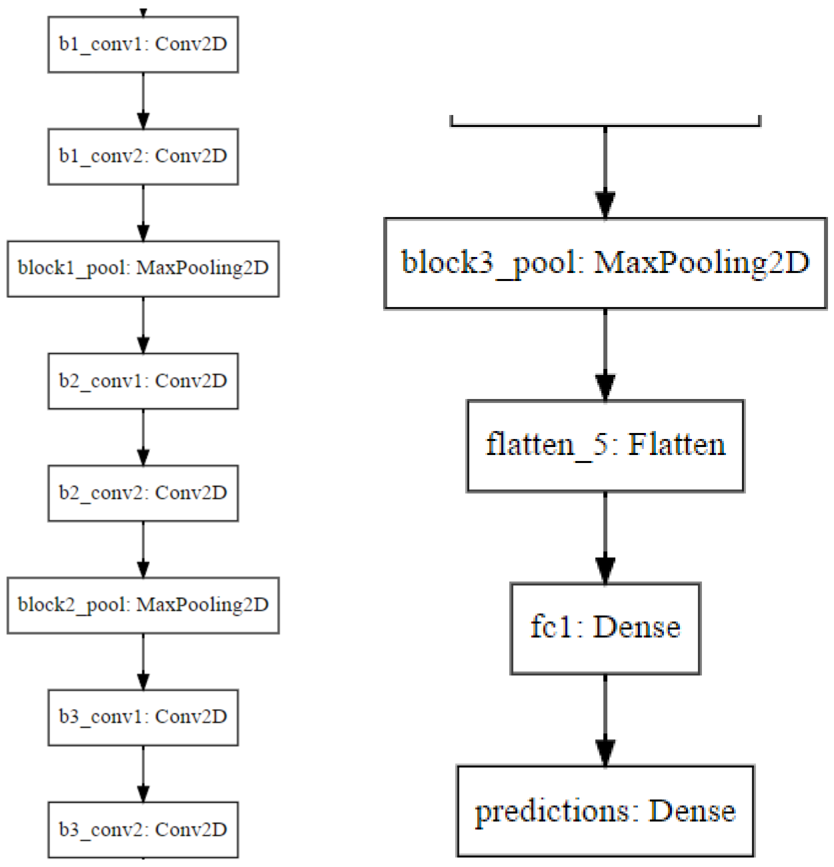
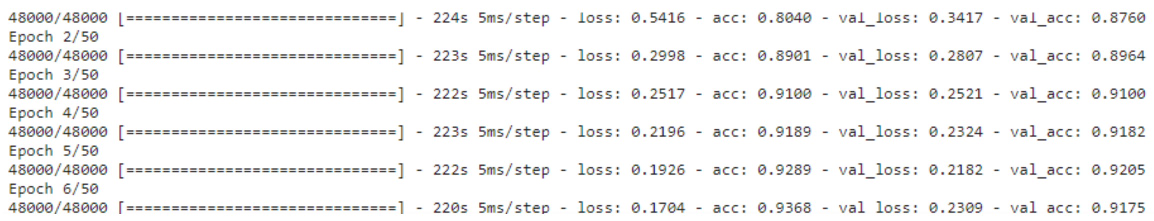
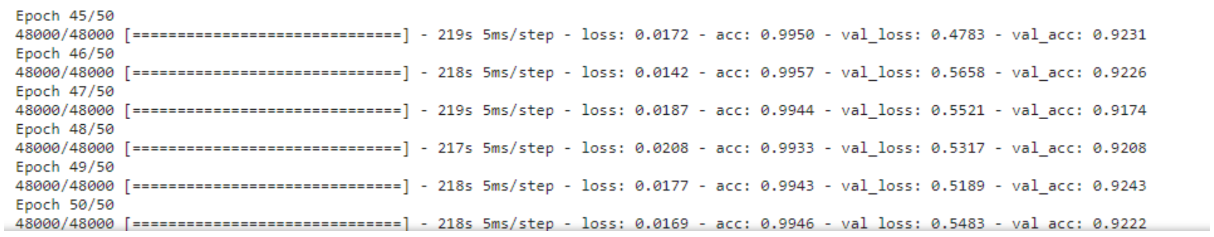


Figure 29 Architecture of used CNN

## 4.5 Model training

The chosen optimizer was the adam and loss function was categorical crossentropy as metric evaluation I decided to only go for accuracy. The accuracy indicates on how well the model correctly predicts the outcome of unseen new data. It is calculated by taking the proportion of correct predictions out of all predictions made by the model.
For the training part the training set were applied to the model. Thee validation set (20 % of the training set) is also used to validate the performance to improve the generalization to unseen data. The initial batch size was set to 128 and a number of epochs was assigned to 50.


```
48000/48000 [==============================] - 224s 5ms/step - loss: 0.5416 - acc: 0.8040 - val_loss: 0.3417 - val_acc: 0.8760
Epoch 2/50
48000/48000 [==============================] - 223s 5ms/step - loss: 0.2998 - acc: 0.8901 - val_loss: 0.2807 - val_acc: 0.8964
Epoch 3/50
48000/48000 [==============================] - 222s 5ms/step - loss: 0.2517 - acc: 0.9100 - val_loss: 0.2521 - val_acc: 0.9100
Epoch 4/50
48000/48000 [==============================] - 223s 5ms/step - loss: 0.2196 - acc: 0.9189 - val_loss: 0.2324 - val_acc: 0.9182
Epoch 5/50
48000/48000 [==============================] - 222s 5ms/step - loss: 0.1926 - acc: 0.9289 - val_loss: 0.2182 - val_acc: 0.9205
Epoch 6/50
48000/48000 [==============================] - 220s 5ms/step - loss: 0.1704 - acc: 0.9368 - val loss: 0.2309 - val acc: 0.9175
```

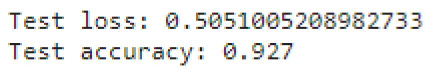


```
Epoch 45/50
48000/48000 [==============================] - 219s 5ms/step - loss: 0.0172 - acc: 0.9950 - val_loss: 0.4783 - val_acc: 0.9231
Epoch 46/50
48000/48000 [==============================] - 218s 5ms/step - loss: 0.0142 - acc: 0.9957 - val_loss: 0.5658 - val_acc: 0.9226
Epoch 47/50
48000/48000 [==============================] - 219s 5ms/step - loss: 0.0187 - acc: 0.9944 - val_loss: 0.5521 - val_acc: 0.9174
Epoch 48/50
48000/48000 [==============================] - 217s 5ms/step - loss: 0.0208 - acc: 0.9933 - val_loss: 0.5317 - val_acc: 0.9208
Epoch 49/50
48000/48000 [==============================] - 218s 5ms/step - loss: 0.0177 - acc: 0.9943 - val_loss: 0.5189 - val_acc: 0.9243
Epoch 50/50
48000/48000 [==============================] - 218s 5ms/step - loss: 0.0169 - acc: 0.9946 - val_loss: 0.5483 - val_acc: 0.9222
```

## Model evaluation without dropout regularization technique

The final test accuracy results after applying the test set to the network provided a 92.7 % correctness.

**Figure 30 Training process of CNN**



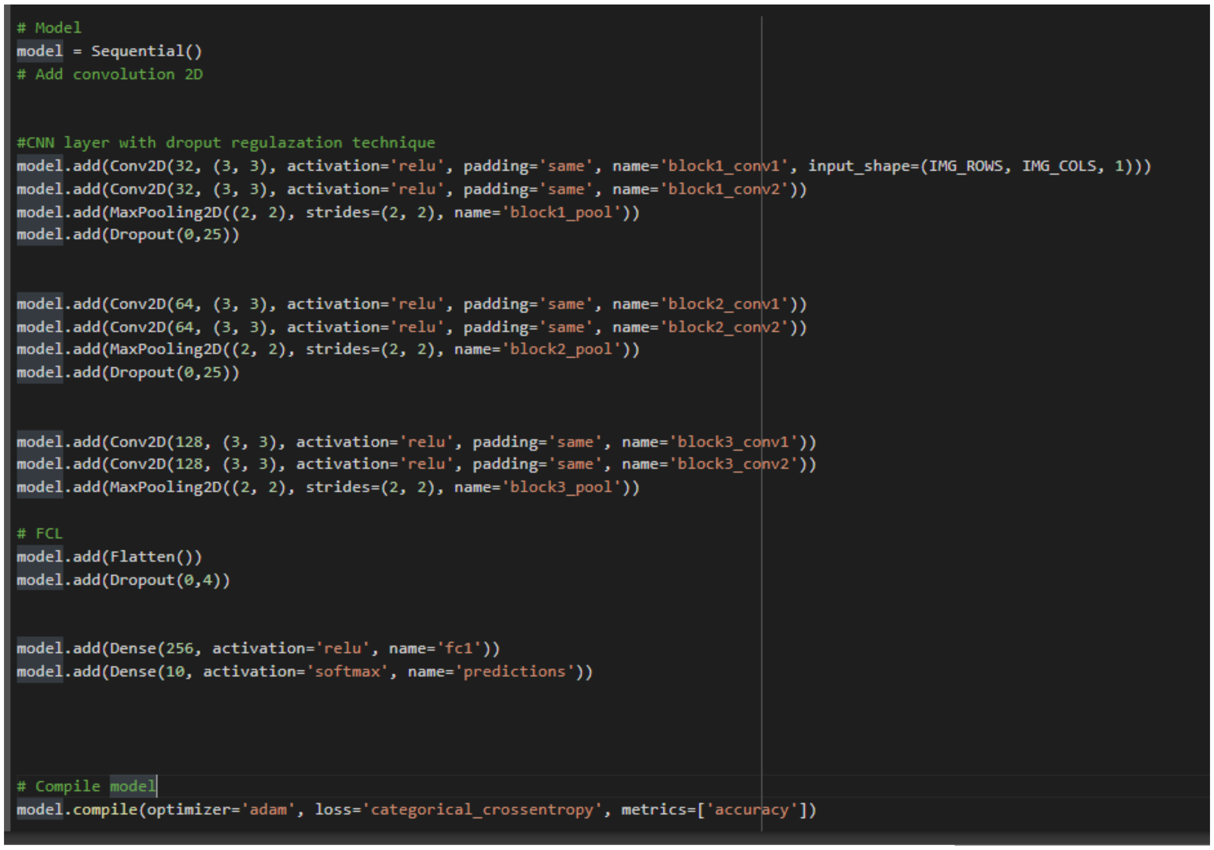
```
Test loss: 0.5051005208982733
Test accuracy: 0.927
```

**Figure 31 Test accuracy on fashion-mnist on trained network**

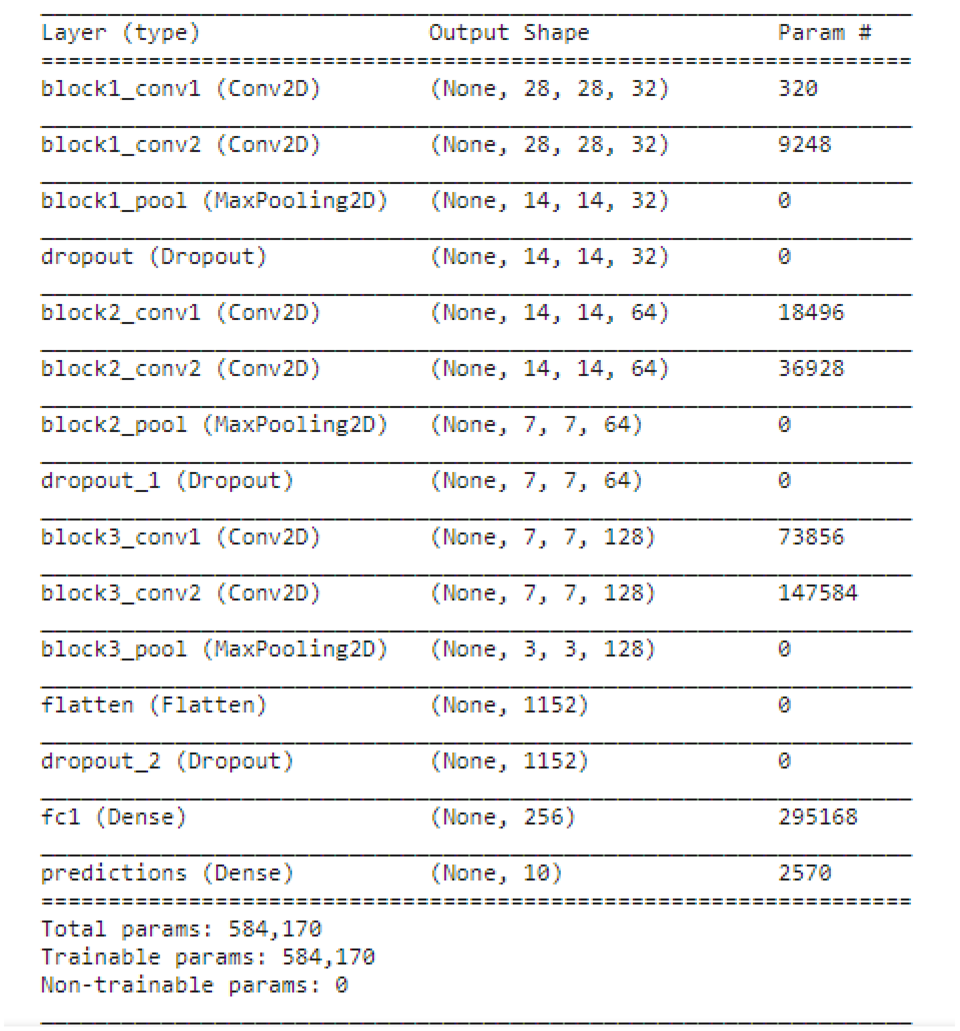
**Figure 32 training and validation accuracy & loss**

## 6.5 Model architecture design with dropout regularization techniques

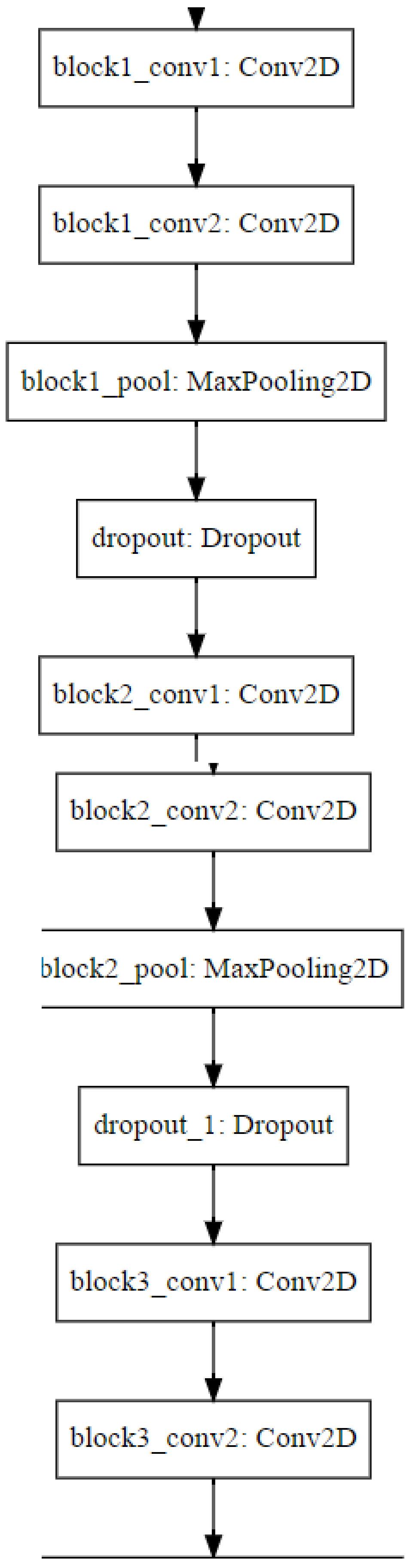
This time the exact same convolutional neural network architecture was used with same hyperparameters and dataset. One minor change is that the dropout regularization technique will be applied at the end of every building block:

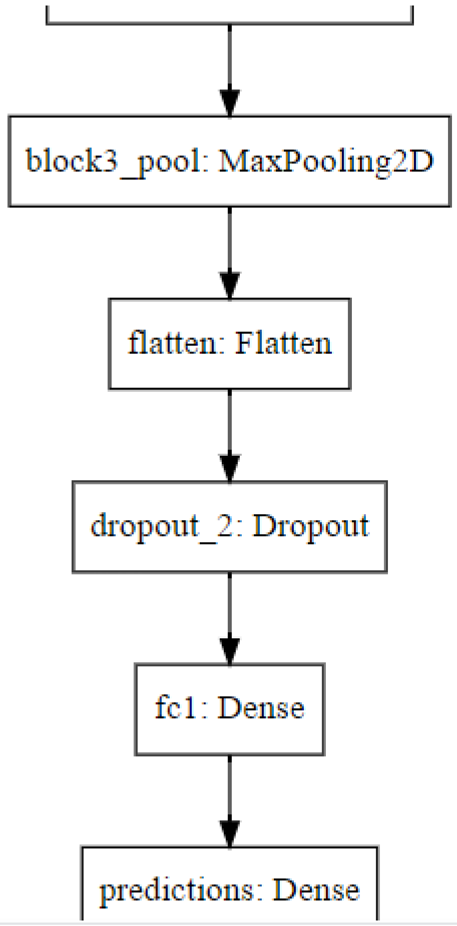


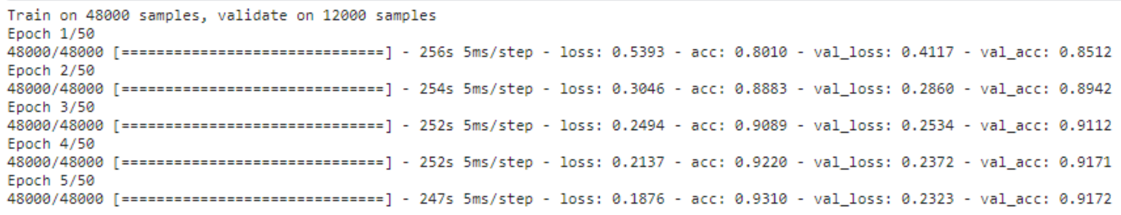
```
# Model
model = Sequential()
# Add convolution 2D


#CNN layer with droput regulazation technique
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', name='block1_conv1', input_shape=(IMG_ROWS, IMG_COLS, 1)))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', name='block1_conv2'))
model.add(MaxPooling2D((2, 2), strides=(2, 2), name='block1_pool'))
model.add(Dropout(0,25))


model.add(Conv2D(64, (3, 3), activation='relu', padding='same', name='block2_conv1'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', name='block2_conv2'))
model.add(MaxPooling2D((2, 2), strides=(2, 2), name='block2_pool'))
model.add(Dropout(0,25))


model.add(Conv2D(128, (3, 3), activation='relu', padding='same', name='block3_conv1'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', name='block3_conv2'))
model.add(MaxPooling2D((2, 2), strides=(2, 2), name='block3_pool'))

# FCL
model.add(Flatten())
model.add(Dropout(0,4))


model.add(Dense(256, activation='relu', name='fc1'))
model.add(Dense(10, activation='softmax', name='predictions'))




# Compile model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```



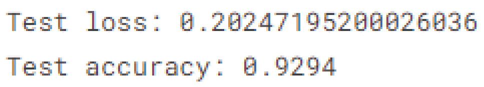
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
block1_conv1 (Conv2D)        (None, 28, 28, 32)        320
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 28, 28, 32)        9248
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 14, 14, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 32)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 14, 14, 64)        18496
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 14, 14, 64)        36928
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 7, 7, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 7, 7, 128)         73856
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 7, 7, 128)         147584
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 3, 3, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 1152)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 1152)              0
_________________________________________________________________
fc1 (Dense)                  (None, 256)               295168
_________________________________________________________________
predictions (Dense)          (None, 10)                2570
=================================================================
Total params: 584,170
Trainable params: 584,170
Non-trainable params: 0
_________________________________________________________________
```


block1_conv1: Conv2D
block1_conv2: Conv2D
block1_pool: MaxPooling2D
dropout: Dropout
block2_conv1: Conv2D
block2_conv2: Conv2D
block2_pool: MaxPooling2D
dropout_1: Dropout
block3_conv1: Conv2D
block3_conv2: Conv2D





```
Train on 48000 samples, validate on 12000 samples
Epoch 1/50
48000/48000 [==============================] - 256s 5ms/step - loss: 0.5393 - acc: 0.8010 - val_loss: 0.4117 - val_acc: 0.8512
Epoch 2/50
48000/48000 [==============================] - 254s 5ms/step - loss: 0.3046 - acc: 0.8883 - val_loss: 0.2860 - val_acc: 0.8942
Epoch 3/50
48000/48000 [==============================] - 252s 5ms/step - loss: 0.2494 - acc: 0.9089 - val_loss: 0.2534 - val_acc: 0.9112
Epoch 4/50
48000/48000 [==============================] - 252s 5ms/step - loss: 0.2137 - acc: 0.9220 - val_loss: 0.2372 - val_acc: 0.9171
Epoch 5/50
48000/48000 [==============================] - 247s 5ms/step - loss: 0.1876 - acc: 0.9310 - val_loss: 0.2323 - val_acc: 0.9172
```



```
Test loss: 0.20247195200026036
Test accuracy: 0.9294
```

## Results & Conclusion

After re-evaluating the network with only one "slight" difference, the dropout technique was implemented the second time. The final results show an improvement in test accuracy since the network accuracy was 92,94 % on the test set when dropout was used. Compared to 92,7  % when none regularization techniques were utilized.

# 7. RECURRENT NEURAL NETWORKS

**Recurrent Neural Networks** (RNNs) are a type of neural network that is particularly well suited for processing sequential data, such as time series or natural language. The key feature of an RNN is that it maintains a hidden state, which is passed from one step to the next and allows the network to remember information from previous steps.

In a traditional feedforward neural network, information flows in one direction, from input to output, with no memory of previous inputs. However, in an RNN, the output of each step is used as an input for the next step, in addition to the input of that step.

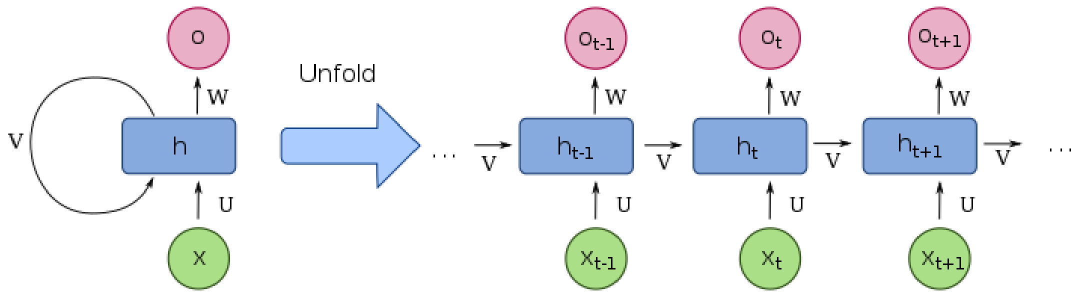


Figure 33 Unrolling of recurrent neural network

This allows the network to maintain a hidden state, which is a vector of activations that encodes information about the previous inputs.

In more detail, an RNN is composed of a series of recurrent units, each of which takes as input both the current input and the hidden state from the previous step. Each recurrent unit applies a non-linear transformation to this input, producing a new hidden state and an output. The hidden state from one recurrent unit is passed as input to the next recurrent unit in the sequence.

RNNs can be unrolled through time to show the flow of information through the network. This unrolled version of the network can be thought of as a deep neural network, with the hidden state (h) acting as the hidden layer.

RNNs are used in a variety of applications, including natural language processing, speech recognition, machine translation, image captioning, and time series forecasting. They have also been used to generate new text, music, and other forms of media.

One of the main challenges with training RNNs is that the gradients that are used to update the weights can vanish or explode as they are backpropagated through time. This problem is known as the vanishing gradient problem and can be addressed using techniques such as gradient clipping or using more advanced variants of RNNs such as LSTM and GRU.

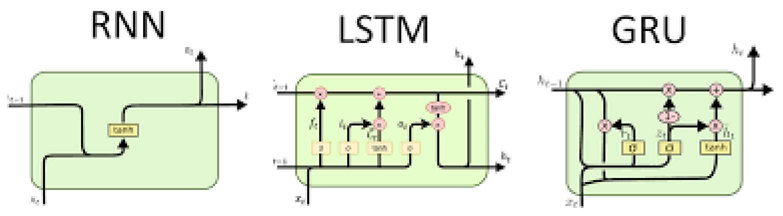


Figure 34 Different kind of RNN's

**LSTM**

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) that is designed to overcome the limitations of traditional RNNs in modeling long-term dependencies in sequential data.

A traditional RNN consists of a single layer of recurrent neurons, which process sequential input data one element at a time, and maintain an internal hidden state that captures the history of the processed elements. However, traditional RNNs have difficulty in maintaining a stable hidden state when the input sequence is very long and/or has complex dependencies between elements, due to the vanishing gradient problem.

LSTMs, on the other hand, have a more complex architecture that allows them to maintain a stable hidden state over longer periods of time. Each LSTM cell contains three gates: the input gate, the forget gate, and the output gate. These gates control the flow of information into and out of the cell, and the flow of information from one cell to the next, allowing the LSTM to selectively retain or discard information from the input sequence.

The input gate controls the flow of new information into the cell, by multiplying the input data by a weight matrix and applying an activation function. The forget gate controls the flow of information out of the cell, by multiplying the previous hidden state by a weight matrix and applying an activation function. The output gate controls the flow of information out of the cell, by multiplying the previous hidden state by a weight matrix and applying an activation function.

In addition to these gates, LSTM cells also have a memory cell, which is used to store information over time. The memory cell is updated at each time step by adding the product of the input gate and the input data to the product of the forget gate and the previous memory cell.

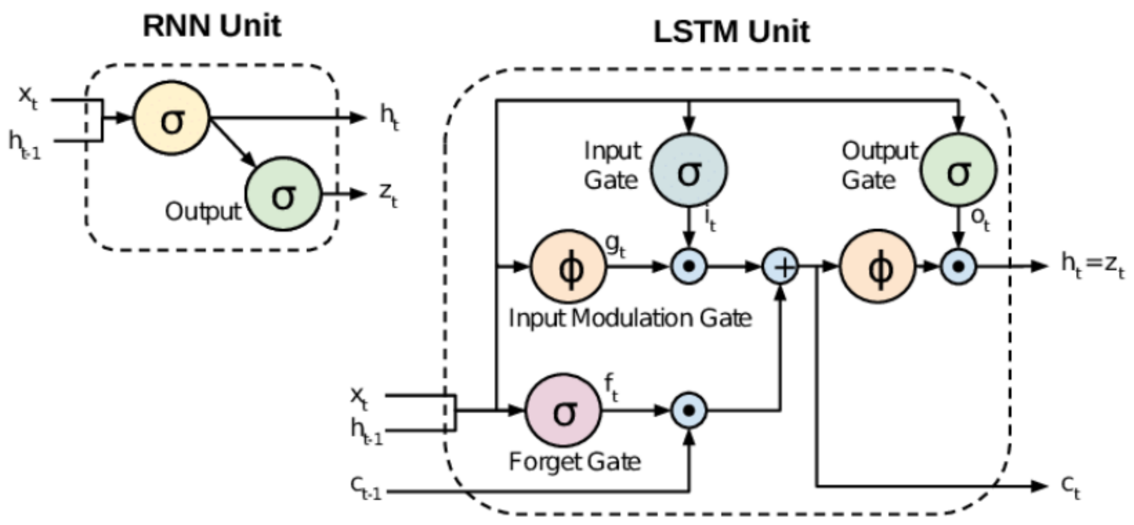


Figure 35 Input , forget and output gates in LSTM

### Attentions

Attention in neural networks refers to a mechanism that allows the model to focus on specific parts of the input when processing it. This is done by weighting different parts of the input according to their importance, and then using these weights to compute a weighted sum of the input. Attention mechanisms have been shown to be particularly effective in tasks such as machine translation and image captioning, where the model needs to be able to selectively focus on different parts of the input. Attention mechanisms can be applied to both the encoder and decoder parts of a neural network.

### Transformers

The Transformer is a neural network architecture that was introduced in the paper "Attention Is All You Need" by Vaswani et al. in 2017 [4]. It is based on the idea of self-attention, which allows the model to weigh the importance of different parts of the input when processing it. The Transformer architecture is commonly used in natural language processing tasks such as machine translation, text summarization, and language modeling.

The key components of the Transformer architecture are the encoder and decoder, which are both made up of multi-head self-attention layers and feed-forward layers. The encoder takes in the input sequence and produces a set of hidden states that capture the information in the input. The decoder then takes these hidden states as input and generates the output sequence.

The multi-head self-attention layers in the Transformer are responsible for computing the attention weights. They work by first projecting the input and the output into a set of queries, keys, and values. These are then used to compute a dot-product attention between the queries and keys, which produces a set of attention weights. These attention weights are then used to compute a weighted sum of the values, which is then used as input to the next layer.

The feed-forward layers in the Transformer are responsible for processing the output from the self-attention layers. They typically consist of a linear layer followed by a non-linear activation function.

One of the key advantages of the Transformer architecture is that it allows for parallel computation of the self-attention layers, which makes it much faster to train than previous architectures such as RNNs and LSTMs. Additionally, the Transformer's architecture allows the model to efficiently process input of any length, making it well-suited to tasks such as machine translation where input and output sequences can have varying lengths.

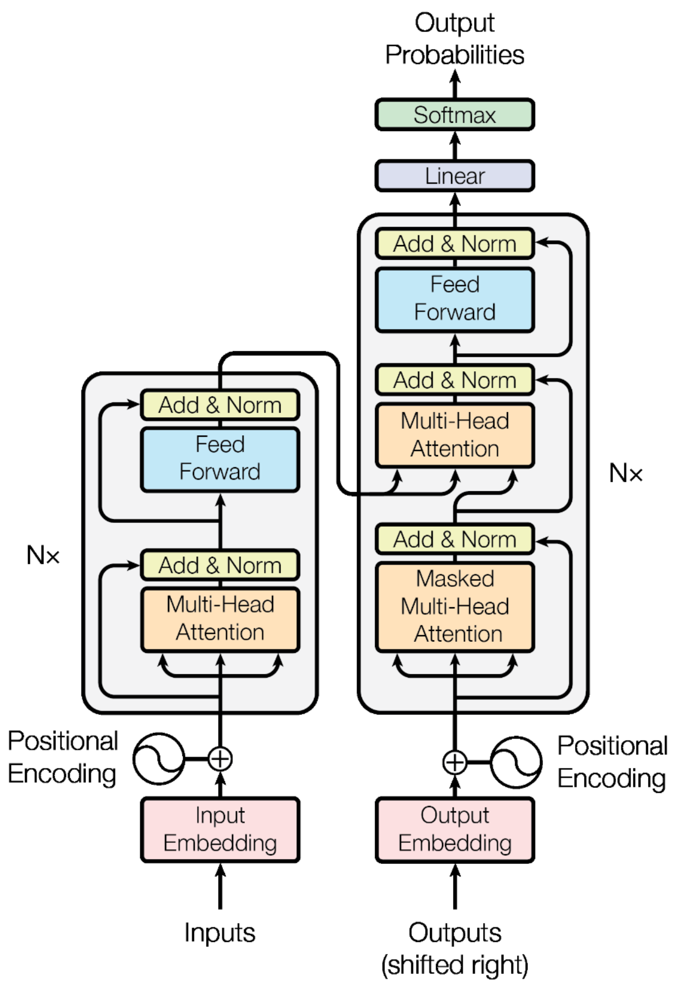


Figure 36 Transformer architecture with encoder and decoder

## 8. LEARNING OUTCOMES AND FINAL CONCLUSIONS

After consuming and learning about deep learning, the presented literature material on the moodle and online. I have received a good amount of knowledge about the topics of different kind of AI concepts,It has led to a deeper understanding on the applications of ANN’s and in real life scenarios. Artificial intelligence is about using an "agent" that interacts with an environment to learn and take actions to maximize its changes to achieve its goals. Machine learning models are trained to predict output based on recorded data. This course's main focus is all about deep neural networks. The vital aspect of having large amount of good data, loss- and activation functions impact on the network. Neuromorphic engineering is the topic of electrical engineering hardware systems.

**Describe the function of formal neuron models and neural network architectures such as**

**feed-forward networks, convolutional networks, and recurrent networks:**
During the course, I've understood the basics and fundamentals of different NN models and networks. I have gained knowledge on how the feed-forward and back-propagation processes are conducted. I have also learnt how the ANNs take advantage of activation functions ,Multi layer perceptrons and on how different kind of loss functions are used. in order to achieve proper outputs throughout the network. In the latter part of the course, I have learned how to "abuse" the CNN and recurrent neural network (RNN) models when it comes to natural language processing. Because RNNs are a great tool when it comes
to having a sequence of input is of an unknown length.The project gave me adeeper understanding of how to implement ANNs and CNNs for image classification.

Demonstrate how neural networks can be trained, validated, and tested with supervised and unsupervised methods and analyze how different hyperparameters and regularization affect model generalization.
Some of the presentations gave me a good introduction to training techniques.
and methods on specific networks. In the project i learned on how to trained
using various datasets and splitting up the data into different datasets. Which gave
give me a better understanding of the difference between hyperparameters such as epochs and learning rates. I now have a better understanding of the difference between supervised and unsupervised learning. Supervised learning Means that the data or input (call input x) is labeled with a output (y) data(x:y).One example of supervised learning is the MNIST digit dataset. When working with the MNIST data set, you train the neural network by finding specific patterns in the input image (x = pixels of a digit image) using optimizers and loss functions. which are calculated depending on the correctness of the
predicted value (), compared to its real labeled value (y). The optimized and adjusted parameters are then validated by a validation set in order to avoid under- or over-fitting.

Give examples of learning machines involving neural network processors and neuromorphic systems and describe their properties and motivations, including ethical considerations:
In the live sessions, we once had a lecture regarding ethics and morals, and I learned about the difficulties in terms of what to do and who to blame if an AI causes a major incident of some sort. some machines involving neural network processors are the GPUs a better gpu

resulting in faster processing computation. One neuromorphic system I have learned about is the TrueNorth, which is designed to have a structure similar to the human brain.

## 9. REFERENCES

[1]

[2]

https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network
**https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53**

Attention Is All You Need

https://www.kaggle.com/datasets/jsphyg/weather-dataset-rattle-package?resource=download

https://www.tensorflow.org/datasets/catalog/fashion_mnist

https://arxiv.org/abs/1706.03762

Convolutional neural networks: an overview and application in radiology

https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9

https://en.wikipedia.org/wiki/Recurrent_neural_network

https://www.ibm.com/topics/recurrent-neural-networks

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks#architecture