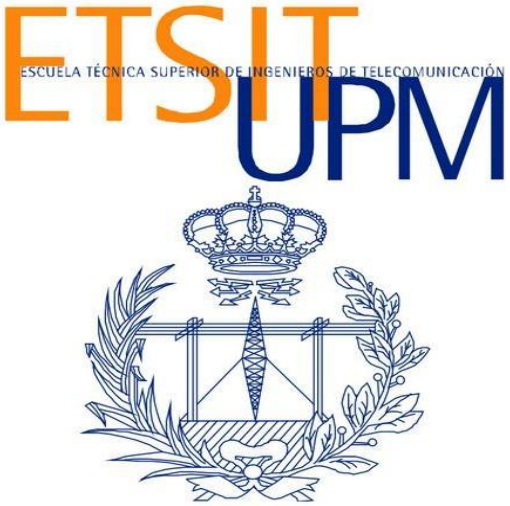


Course:

Predictive and Descriptive Learning

Final Deep-learning Report

The main objective of this Report/Presentation is to demonstrate your abilities to design and develop a Machine Learning models to address a particular use case.

Name: (Olle Elvland)

Date: ……..

CONTENTS

1. DEVELOPING MACHINE LEARNING MODELS..... FEL! BOKMÄRKET ÄR INTE DEFINIERAT.

1.1. Problem description.....4

1.2. Machine Learning approach..... **Fel! Bokmärket är inte definierat.**

2. DATA DESCRIPTION ..... FEL! BOKMÄRKET ÄR INTE DEFINIERAT.

2.1. Data bases..... **Fel! Bokmärket är inte definierat.**

2.2. Data preparation ..... **Fel! Bokmärket är inte definierat.**

3. EXPLORATORY DATA ANALYSIS (EDA). FEL! BOKMÄRKET ÄR INTE DEFINIERAT.

3.1. Introduction ..... **Fel! Bokmärket är inte definierat.**

3.2. Techniques ..... **Fel! Bokmärket är inte definierat.**

3.3. Results & Conclusions ..... **Fel! Bokmärket är inte definierat.**

4. EXPERIMENTAL SETUP..... FEL! BOKMÄRKET ÄR INTE DEFINIERAT.

4.1. Introduction ..... **Fel! Bokmärket är inte definierat.**

5. MACHINE LEARNING MODELS..... FEL! BOKMÄRKET ÄR INTE DEFINIERAT.

5.1. Model selection ..... **Fel! Bokmärket är inte definierat.**

6. RESULTS..... FEL! BOKMÄRKET ÄR INTE DEFINIERAT.

6.1. Evaluation methodology ..... **Fel! Bokmärket är inte definierat.**

6.2. Discussion ..... **Fel! Bokmärket är inte definierat.**

7. CONCLUSIONS..... FEL! BOKMÄRKET ÄR INTE DEFINIERAT.

8. REFERENCES..... 29

APPENDIX A: SELF-SUPERVISED LEARNING . FEL! BOKMÄRKET ÄR INTE DEFINIERAT.

LIST OF TABLES

Table 1. OSA dataset content **.....** Fel! Bokmärket är inte definierat.

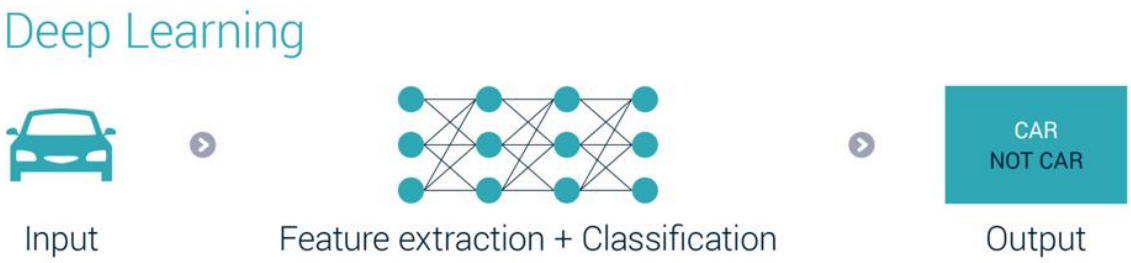
LIST OF FIGURES

Figure 1. Sleep Apnea (from [2]) **.....** Fel! Bokmärket är inte definierat.

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.



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 mini projects based on the basic DL models. 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 with multiple layer for weather prediction
2) A **Convolution neural network** to classify different cloth images using the fashion MNIST dataset.

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 and R 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.

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) Raining prediction with use of 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.

To differentiate syntax code and normal text, I've emphasized the corresponding syntax representing by highlighting the syntax code for python text either as an image or bold text.

4.3 code description

Initially some packages and libraries were imported from tensorflow library :

```
from tensorflow import keras
```

, Keras were import for building the neural networks.

The building of the model was generated through from keras built in **sequential** model which is a linear stack of various layers for neural networks. The layers building block implementations like Dense and input was also imported from tenserflow keras layers :

```
from keras.models import Sequential
from keras import Input
from keras.layers import Dense
```

Pandas and numpy were implemented for data manipulation:

```
import pandas as pd
import numpy as np
```

Sklearn library for model evaluation , data split utilization for test and train set and metricses:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

For visual presntation plotly library was used :

```
import plotly
import plotly.express as px
import plotly.graph_objects as go
```

4.2.1 Data description

The following dataset used in the project is 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 Austrailian weather stations. Since the goal of the given assignment was mainly focusing on learning and not to 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 and weather locations at 3pm

```
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
```

- The **RainTomorrow** parameter describes whether it rained or not the following day

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

For reading the WeaterAUS.csv file, the dataset was converted into a dataframe assigned to df using pandas read_csv :

```
df=pd.read_csv('/content/weatherAUS.csv', encoding='utf-8')
```

4.2 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 droped all the cloumns where **Raintomorrow**=NAN values:

```
df=df[pd.isnull(df['RainTomorrow'])==False]
```

Also where other column values where missing values , column values got filled with the mean value with the corresponding columns using :

```
df=df.fillna(df.mean())
```

To change the datatype (instead of object datatype like “yes” and “no”, a flag was set to 1 if it was raining tomorrow/today and a 0 if it didn’t rain) of target variable a flag for raintoday and **raintomorrowflag** was made using lambda notation:

```
df['RainTodayFlag']=df['RainToday'].apply(lambda x: 1 if x=='Yes' else 0)
df['RainTomorrowFlag']=df['RainTomorrow'].apply(lambda x: 1 if x=='Yes' else 0)
```

```
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 1 Before flagging the 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 2 After flagging the data

A snapshot of the whole dataset:

`df`:

```
df
```

```
<ipython-input-3-7fa061261a8d>:11: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid
  df=df.fillna(df.mean())
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am	Pressure3pm	Clou
0	2008-12-01	Albury	13.4	22.9	0.6	5.469824	7.624853	W	44.0	W	WNW	20.0	24.0	71.0	22.0	1007.7	1007.1	8.000
1	2008-12-02	Albury	7.4	25.1	0.0	5.469824	7.624853	WNW	44.0	NNW	WSW	4.0	22.0	44.0	25.0	1010.6	1007.8	4.437
2	2008-12-03	Albury	12.9	25.7	0.0	5.469824	7.624853	WSW	46.0	W	WSW	19.0	26.0	38.0	30.0	1007.6	1008.7	4.437
3	2008-12-04	Albury	9.2	28.0	0.0	5.469824	7.624853	NE	24.0	SE	E	11.0	9.0	45.0	16.0	1017.6	1012.8	4.437
4	2008-12-05	Albury	17.5	32.3	1.0	5.469824	7.624853	W	41.0	ENE	NW	7.0	20.0	82.0	33.0	1010.8	1006.0	7.000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
145454	2017-06-20	Uluru	3.5	21.8	0.0	5.469824	7.624853	E	31.0	ESE	E	15.0	13.0	59.0	27.0	1024.7	1021.2	4.437
145455	2017-06-21	Uluru	2.8	23.4	0.0	5.469824	7.624853	E	31.0	SE	ENE	13.0	11.0	51.0	24.0	1024.6	1020.3	4.437
145456	2017-06-22	Uluru	3.6	25.3	0.0	5.469824	7.624853	NNW	22.0	SE	N	13.0	9.0	56.0	21.0	1023.5	1019.1	4.437
145457	2017-06-23	Uluru	5.4	26.9	0.0	5.469824	7.624853	N	37.0	SE	WNW	9.0	9.0	53.0	24.0	1021.0	1016.8	4.437
145458	2017-06-24	Uluru	7.8	27.0	0.0	5.469824	7.624853	SE	28.0	SSE	N	13.0	7.0	51.0	24.0	1019.4	1016.5	3.000

Model preparation:

Step 1 extracting features:

Since we are only interested in the **humidity3pm** as feature and **RainTomorrowflag** as target variable, the feature variable (X) assigned to an X variable and the **RainTomorrowflag** as **(**y):

```
X=df[['Humidity3pm']]
y=df['RainTomorrowFlag'].values

print(X)
print(y)
```

```
        Humidity3pm
0              22.0
1              25.0
2              30.0
3              16.0
4              33.0
...             ...
145454         27.0
145455         24.0
145456         21.0
145457         24.0
145458         24.0

[142193 rows x 1 columns]
[0 0 0 ... 0 0 0]
```

Step 2 splitting dataset:

In order to split up the test and train data following code line was written:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

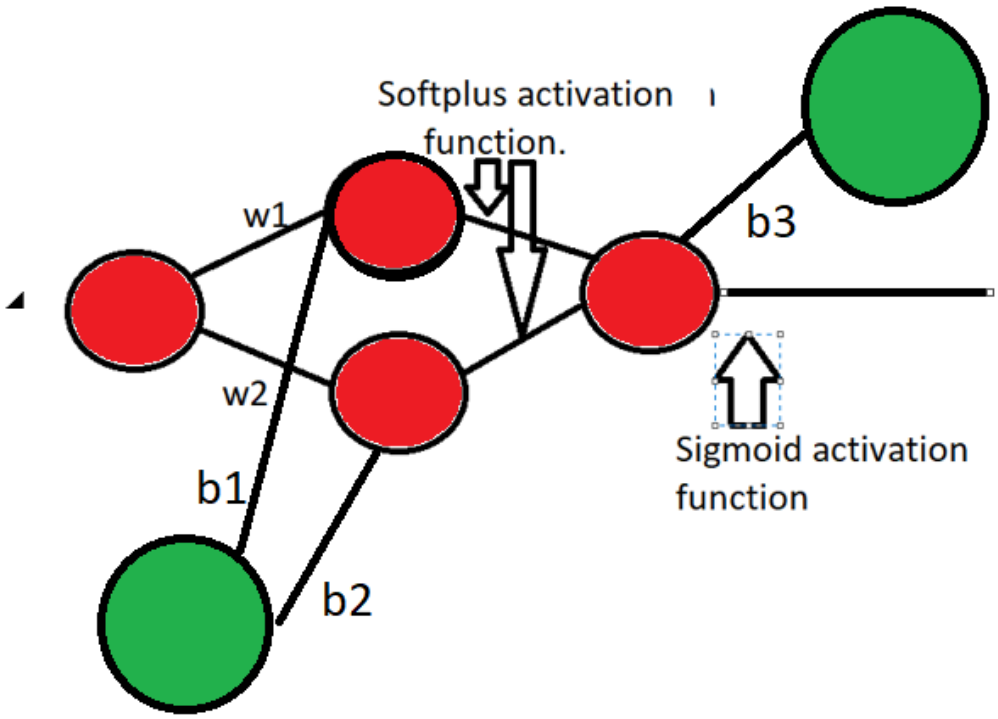
The train_test_split function from scikit-learn library splits up desired percentage of random testing and training samples. 80 % of the feature and labels are split into the training data to fit the model (X_train and y_train), while 20 % (X_test and Y_test) is unseen and untrained data used to validate the model.

Step 3 model structure:

The building of the model was generated through from keras built in **sequential** model function which is built into keras and is a linear stack of various layers for neural networks:

```
model = Sequential(name="Model-One-Input")
model.add(Input(shape=(1,), name='Input-Layer'))
model.add(Dense(2, activation='softplus', name='Hidden-Layer'))
model.add(Dense(1, activation='sigmoid', name='Output-Layer'))
```

These four simple lines of code generates a sequential “deep” feed forward neural network model named “Model-one-input”. The input layer is always at the beginning of the network and is therefore added to the model primarly. The input layer only consist of 1 input neuron. The next added layer is a dense hidden layer consisting of 2 neurons and is called Hidden layer with a softplus activation function applied to it at the end. The final layer is another dense layer but is used as output layer with only one neuron with an sigmoid activation function at the end.



Step 4 compile model

In Keras, the compile method is used to configure the learning process before training a model. It takes three main arguments: the optimizer, loss function and metrics

```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['Accuracy', 'Precision', 'Recall'],
             )
```

In this case the adam optimizer was selected with binary_crossentropy as loss function and accuracy, precision and recall as metrics.

Step 5 fitting/training the data

To fit the training data and train the model the built in fit function was used. The variables was **X_train** = input data, **y_train** = train data, **batch size** = the number of training examples used in each iteration of the training process. **Epochs** = The number of times the entire training dataset is passed through the network during training. And **validation_split =** percentage of the training set dedicated for validation set.

```
model.fit(X_train,
          y_train,
          batch_size=10,
          epochs=3,
          validation_split=0.2,

         )
```

Step 6 Model predictions

The prediction model is used for computing the trained models performance on the train and test data. As the output layer utilized a softmax function of the end which provides percentage based outcomes. The predicted class labels on training data was right if output was bigger than 50 % . Which indicates that it would much likely rain the next day and would therefore be set to a value of 1, else 0.

```
pred_labels_tr = (model.predict(X_train) > 0.5).astype(int)

pred_labels_te = (model.predict(X_test) > 0.5).astype(int)
```

```
9101/9101 [==============================] - 29s 3ms/step - loss: 0.7071 - Accuracy: 0.6750 - precision: 0.33
Epoch 2/3
9101/9101 [==============================] - 27s 3ms/step - loss: 0.2109 - Accuracy: 0.7712 - precision: 0.49
Epoch 3/3
9101/9101 [==============================] - 31s 3ms/step - loss: 0.2109 - Accuracy: 0.7714 - precision: 0.49
3555/3555 [==============================] - 7s 2ms/step
889/889 [==============================] - 1s 1ms/step
```

Step 7 model & performance summary

The summary function provides a description of the network architecture:

```
model.summary()
```

Code snippet above shows us the network summary of the model given 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

=================================================================
```

By iteratively travers through the layers of the network you may give insight of the final layer names (layer.name), the weights of corresponding layer (layer.get_weigths()[0]) and the biases using (layer.get_weights()[1] function)

```
for layer in model.layers:
    print("Layer: ", layer.name)
    print("  --Kernels (Weights): ", layer.get_weights()[0])
    print("  --Biases: ", layer.get_weights()[1])
```

Code snippet above

```
-------------------- Weights and Biases --------------------
Layer:  Hidden-Layer
  --Kernels (Weights):  [[-0.0165867  0.3791127]]
  --Biases:  [ 1.9618193 -1.0385997]
Layer:  Output-Layer
  --Kernels (Weights):  [[-1.774521  ]
 [ 0.10922126]]
  --Biases:  [-0.1752061]
```

Final evaluation on the training and test data may be given using following code snippet :

```
print("")
print('---------- Evaluation on Training Data ----------')
print(classification_report(y_train, pred_labels_tr))
print("")

print('---------- Evaluation on Test Data ----------')
print(classification_report(y_test, pred_labels_te))
print("")
```

The output of classification_report with comparison of actual labels vs predicted give following matrix :

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

           0       0.89      0.79      0.83     88249
           1       0.47      0.65      0.54     25505

    accuracy                           0.75    113754
   macro avg       0.68      0.72      0.69    113754
weighted avg       0.79      0.75      0.77    113754


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

           0       0.89      0.79      0.84     22067
           1       0.47      0.66      0.55      6372

    accuracy                           0.76     28439
   macro avg       0.68      0.72      0.69     28439
weighted avg       0.80      0.76      0.77     28439
```

4. Project 2) image classification with a convolutional neural network on Fashion-MNIST dataset

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 is composed of multiple layers, including :

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

6.1 Project description

In this minor project I decided to with the of use a modification and simplification of one the most famous and successful convolution neural network model architecture: the VGG 16 CNN. With the goal to build and apply 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.

Project 2) code

Similarly to project one, the first thing that had be implemented was some packages and libraries imported from tensorflow library :

```
from tensorflow import keras
```

, Keras is used and imported for building the neural networks.

The building of the CNN model was generated through from keras built in **sequential** model which is a linear stack of various layers for neural networks. The layers building blocks implemented and used for building a CNN in this project is Dense, Flatten , convolutional, dropout and maxpooling, which are very common building blocks within CNN's. These were imported using following code from keras:

```
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense, Flatten, Conv2D, Dropout, MaxPooling2D
```

Pandas and numpy were implemented for data manipulation:

```
import pandas as pd
import numpy as np
```

Sklearn library for model evaluation , data split utilization for test and train set and metricses:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

For visual presentation plotly library was used :

```
import plotly
import plotly.express as px
import plotly.graph_objects as go
```

Initially the train and test-data was read and assigned into a train_data and test_datavariable , using pandas built in read_csv which returns a data frame of a comma separated values file:

```
train_data = pd.read_csv('/content/fashion-mnist_train.csv')
test_data = pd.read_csv('/content/fashion-mnist_test.csv')
```

Some parameters were set for preparing the fashion dataset be suitable for the project :

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

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.

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.

Visualization of the total numbers of training and test images:

```
print("Fashion MNIST train -  rows:",train_data.shape[0]," columns:", train_data.shape[1])
print("Fashion MNIST test -  rows:",test_data.shape[0]," columns:", test_data.shape[1])
```

```
Fashion MNIST train -  rows: 60000  columns: 785
Fashion MNIST test -  rows: 10000  columns: 785
```

In order to visualize the distribution of the classes labels a small get_classes_distribution function was created :

```
def get_classes_distribution(data):

    label_counts = data["label"].value_counts()
    total_samples = len(data)

    for i in range(len(label_counts)):
        label = labels[label_counts.index[i]]
        count = label_counts.values[i]
        percent = (count / total_samples) * 100
        print("{:<20s}:   {} or {}%".format(label, count, percent))

get_classes_distribution(train_data)
```

```
Pullover            :   6000 or 10.0%
Ankle Boot          :   6000 or 10.0%
Shirt               :   6000 or 10.0%
T-shirt/top         :   6000 or 10.0%
Dress               :   6000 or 10.0%
Coat                :   6000 or 10.0%
Sandal              :   6000 or 10.0%
Bag                 :   6000 or 10.0%
Sneaker             :   6000 or 10.0%
Trouser             :   6000 or 10.0%
```

Data['label'].value_counts is describing total number of occurrences of each class (label number) within the train_data dataframe.

Data preprocessing

Some data preprocessing had to be ma de for the future model.

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).

```
def data_preprocessing(raw):
    out_y = keras.utils.to_categorical(raw.label, NUM_CLASSES)
    num_images = raw.shape[0]
    x_as_array = raw.values[:,1:]
    x_shaped_array = x_as_array.reshape(num_images, IMG_ROWS, IMG_COLS, 1)
    out_x = x_shaped_array / 255
    return out_x, out_y
```

Also a separation of the target label as a unique vector.´ The dataset were also split into three different subsets:

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

- 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)
```

6.4 Training the model :

A linear stock of layers with initiated by the Sequential() function.

The .add() method add the layers sequential to eachother. By creating a **convolutional layer(s)** 2D layer using the conv2d() command. The first parameter tells the amount of filtering used in the layer. In the first block, 32 filters were used, in the second, 64 filters were utilized, and in the final block, 128 filters were applied. The following parameter is the kernel or filter size. In this network, all convolution blocks have a filter size of 3 x 3. The next variable is the desired activation function to be applied. In the following structure, all convolution building blocks use the rectified linear unit function to implement non-linearity in the model. The padding parameter was all set to the same padding to preserve the spatial dimensions of the input feature map.

The **Maxpooling layers** are implemented using the MaxPooling2d() function. The first parameter within the method decides the size of the pool (2, 2) to downscale both directions. Stride is the same parameter as in the convolution layer's (2,2) size.

The fully connected layer is starting with a **flatten layer (**flatten()), since all the features have to be flattened so they may be correctly dimensioned and processed within the fully connected NN layer.

After **the flatten layer**, two dense layers were added using the Dense() method. The first dense layer has an input layer of 256 units and a ReLU activation function added at the output. The final one is considered the output layer and has a total of 10 neurons. One neuron for each label class from the fashion-MNIST dataset As standard for a multiclass

classification problem, the softmax function is applied.

```
# Model
model = Sequential()



#Convolution layers
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(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(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(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'])
```

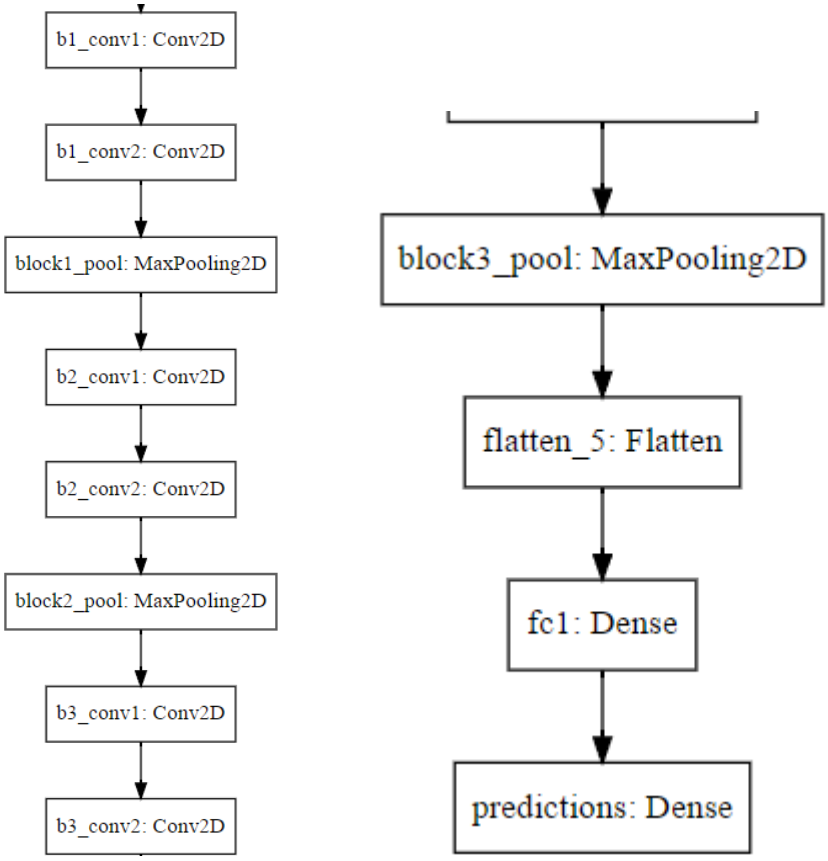
To get a good visual summary of the model following command was used:

`model.summary()`**.** This method provides a summary of all the blocks and layers shape and also the number of parameters within each layer.

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
```

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



For training the model.fit() method was called with parameters such as the split up training and dataset ,hyperparametrs such as batch size and number of epochs, and a finally a validation set (subset of the training set)

```
train_model = model.fit(X_train, y_train,
                  batch_size=BATCH_SIZE,
                  epochs=NO_EPOCHS,
                  verbose=1,
                  validation_data=(X_val, y_val))
```

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
```

For evaluating the test set on the trained network , the model.evaluate method is used. With parameters of the split up test set consisting of 10000 images who has not been seen previously from the network. printing and testing model test loss and accuracy may be executed using following code :

```
core = model.evaluate(X_test, y_test, verbose=0)
rint('Test loss:', score[0])
rint('Test accuracy:', score[1])
```

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

Another training was performed with the dropout regularization technique. It may simply be added to the sequential model using Dropout(x) where x is a value of 1-0 based on perecentage of neuron drop to prevent overfitting :

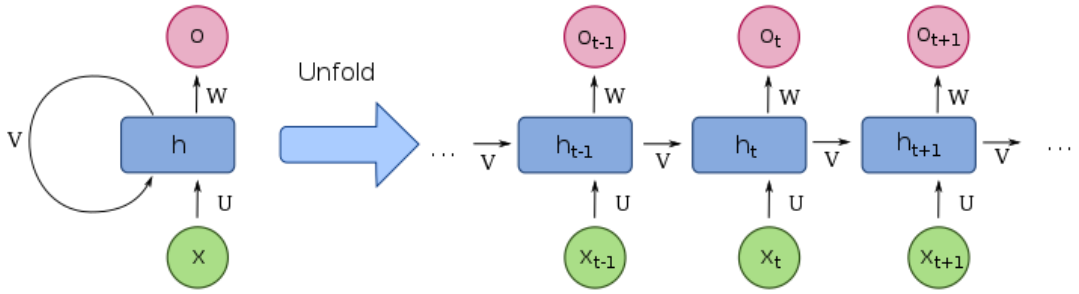
```
# Model
model = Sequential()
# Add convolution 2D


#Convolution ,ppoling and dropout layer
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'])
```

This improved the result slightly on the evaluation score.

5. 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.



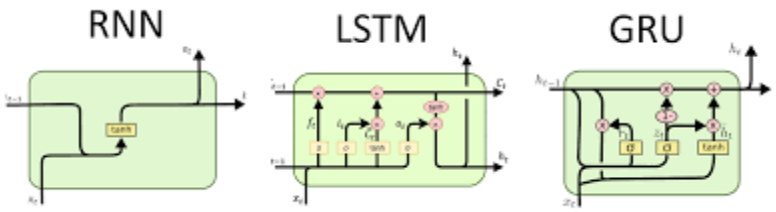
This allows the network to maintain a hidden state, which is a vector of activations that encodes information about the previous inputs.

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.



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.

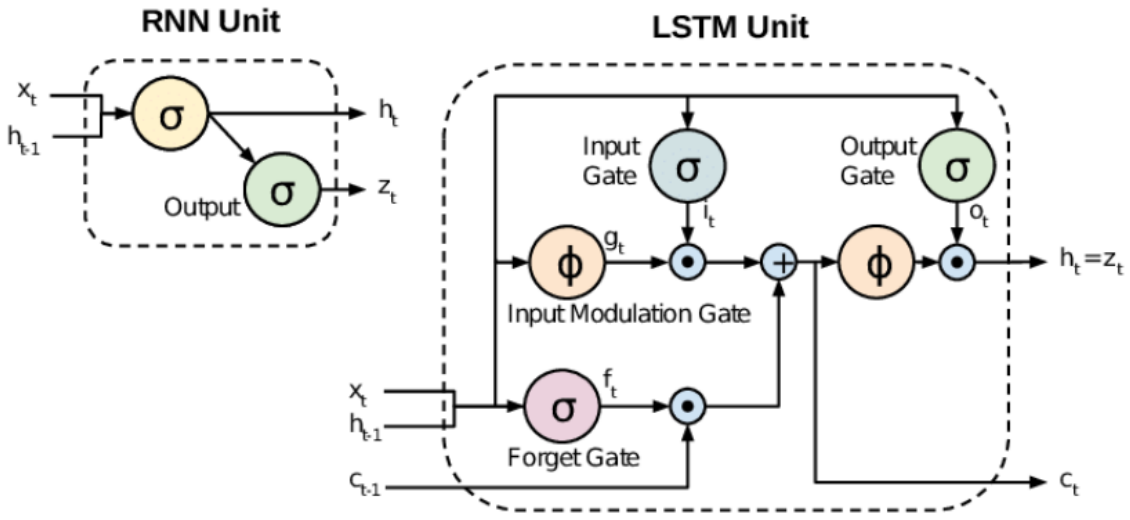
A traditional RNN consists of a single layer of recurrent neurons, which process sequential

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.



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

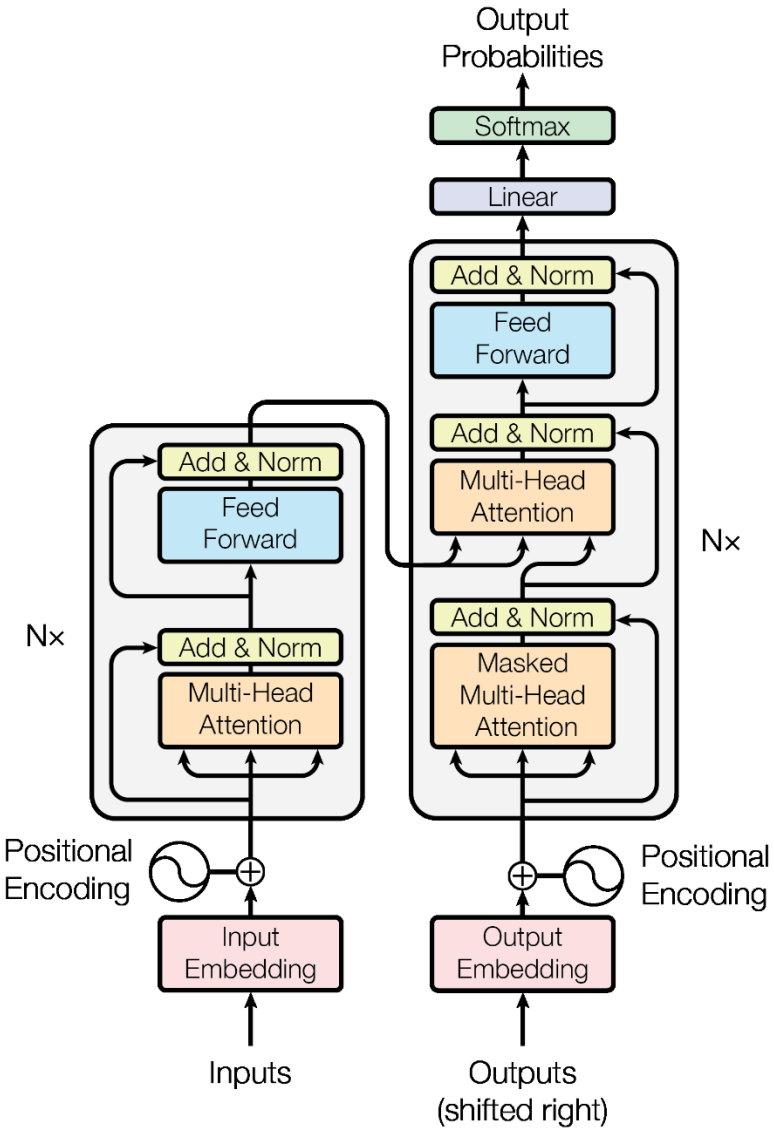
The Transformer is a neural network architecture that was introduced in the paper "Attention Is All You Need" by Vaswani et al. in 2017 [4]. It is based on the idea of self-attention, which allows the model to weigh the importance of different parts of the input when processing it. The Transformer architecture 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.



6. REFERENCES

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

https://d2l.ai/chapter_computer-vision/fine-tuning.html