**Report**

For this project I used keras from tensorflow. For each step I’ll explain my codes step by step.

**Data understanding:**

First things first we need to load the dataset. Luckily MNIST dataset is fairly easy to load. These variables are the initial dataset .

train\_images: (60000, 28, 28)

train\_labels: (60000,)

test\_images: (10000, 28, 28)

test\_labels: (10000,)

**Preprocessing:**

After normalizing the images by dividing them to 255 (which seals the values between 0 and 1), as instructed I changes the shape of the dataset. Now we have 784 pixiels in a vector which makes them features.

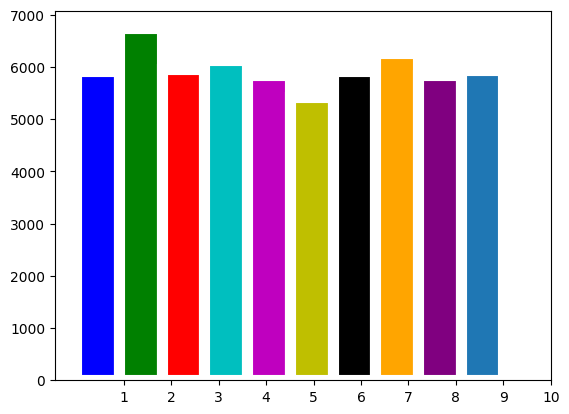
astype changes the value types in the data to float32 in case they have other types.

**Data understanding:**

matplotlib is pretty easy to use so I plotted 5 random images from the train set. Randint should randomly give an integer from 0 to 60000 which is the number of images in the train set . Notice that for this exercise I used the original dataset. Then I index the random image with a color map.

To use histograms in different colors for each class( number ) I had to use a list of colors from Colormap. The bins variable represents the edge of each column (bar) or bin in the histogram.

Patches are the actual bars which are being plotted in the for loop using set\_facecolor which takes the colors from the list I explained earlier. It took a bit of effort but here’s the histogram!!

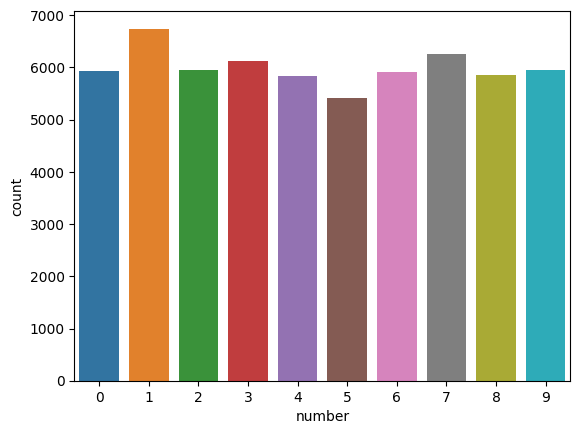


set\_xticks is supposed to add numbers 0 to 9 in the x axis. Without it only the odd numbers would be visible!!!

I looked for other solutions to make a histogram like this and I found one using the seaborn library.

I also made a histogram for comparing test and train sets. In order for the

Countplot to work I had to provide it with a column of a dataframe but I only had a numpy array. So I made a dataframe using pandas and then plotted the column I named number . The new histogram was much better than the previous one:



For the labels, I used one hot encoding to prepare them for the models. Since the the labels are vectors of numbers from 0 to 9 without any classification I had to separate the classes using to\_categorical function.

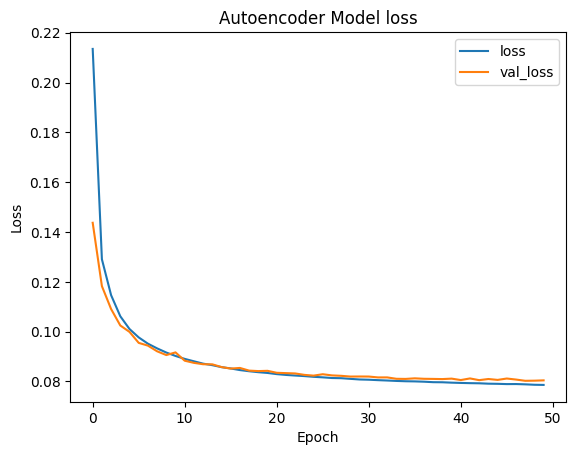
**Autoencoder Model:**

For the autoencoder model I used the model instructed for this project. The input is the normalized data and it has the shape 784 the output will be the processed vectors ( images) which later will be given to the classification model. The inputs and outputs as you can see in the fit function are the images and labels are not needed here since we are only processing and filtering the pixels.

Because of the shape of the output I treated this model as a binary classification which resulted in using the sigmoid activation function. I simple terms the model will indicate how much a pixel will effect the classification model later on.

The adam optimizer is very effective most of the time but I tried other optimizers and the loss increased so I decided to go with adam.

After the model was trained this was the conclusion of loss\_epoch :



There’s no need for accuracy since we are not solving a problem here, we are simply preparing for one!

Shape of decoded\_imgs: (60000, 784)

Shape of train\_labels: (60000,)

Number of samples in decoded\_imgs: 60000

Number of samples in train\_labels: 60000

(60000, 784)

Before I made the classification model I made sure the shapes were in order since shape errors are the most common in machine learning .

I plotted the conclusion . There are small differences between the 2 rows! The top one is the original and the bottom one is the decoded images.

**Classification Model:**

To use the transformed images from Auto encoder I predicted the labels for both train set and test set and later I gave them to the classification model .

Model2 is doing the classification and it’s a bit complex for such a data so at first it was bound to overfit! Using dropout layers and by reducing the number of neurons in each layer I fixed the issue.

Callbacks are meant to prevent overfitting as well and they enhanced the performance of the model allowing me to use more epochs before the data runs out!! The data running out problem was solved using dropout layers as well .

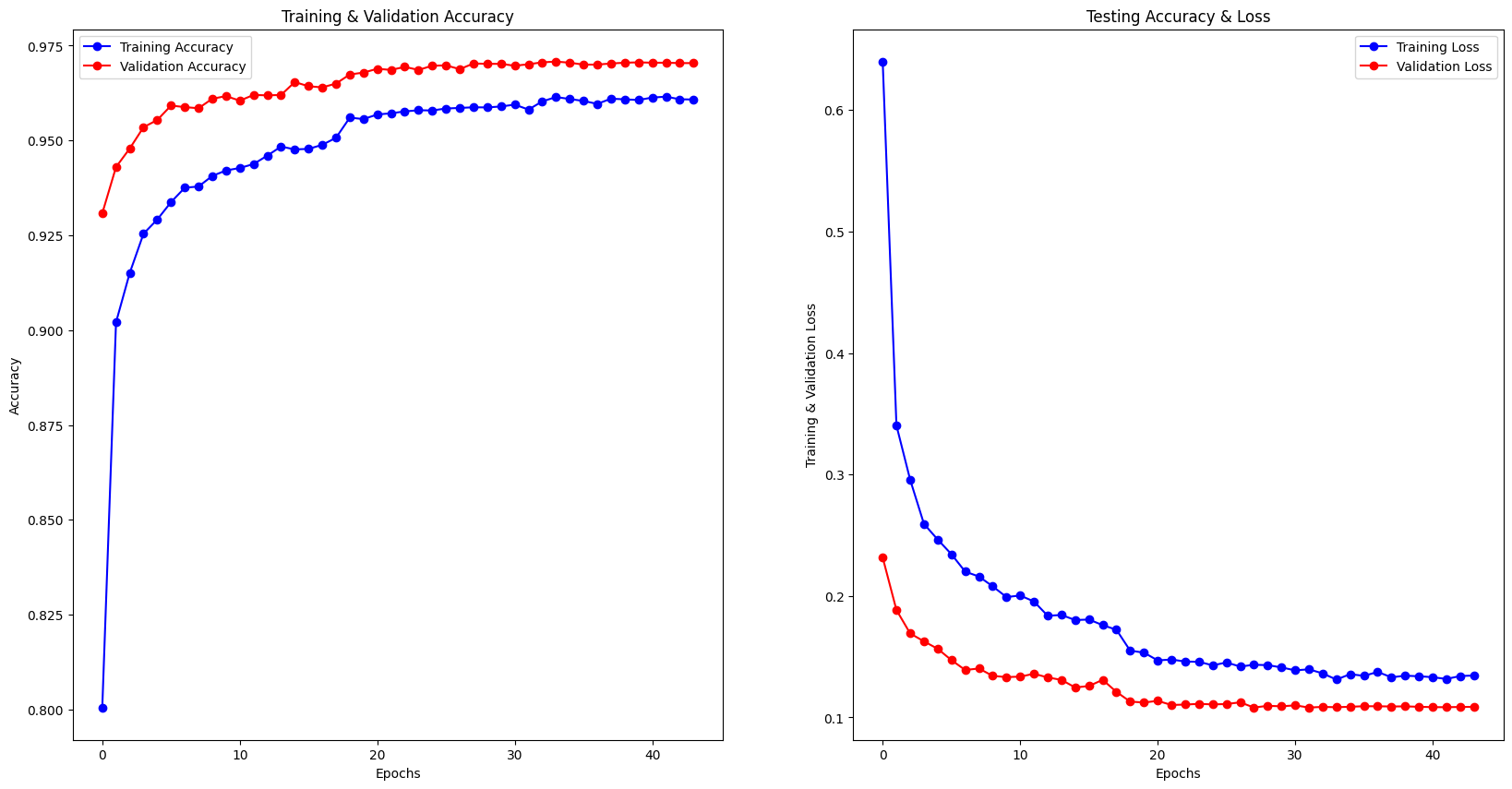
Tip: I used relu function in all my layers since it was recommended in the paper of the assignment and it is the best due to the elimination of the negative values.

These lines are meant to prevent lack of data as well! They helped the training process.

train\_steps = ceil(X\_train.shape[0] / batch\_size)

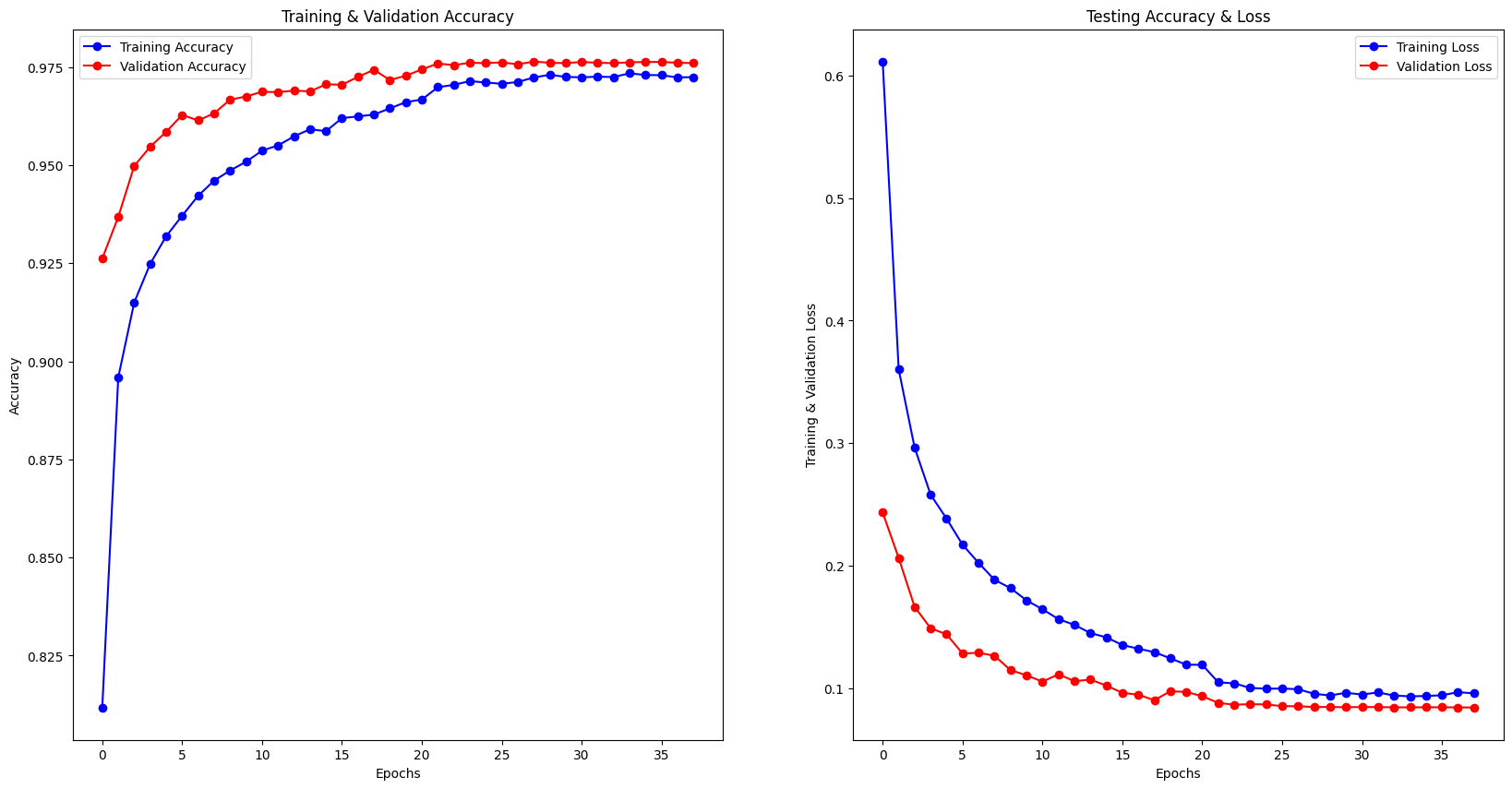
valid\_steps = ceil(X\_test.shape[0] / batch\_size)

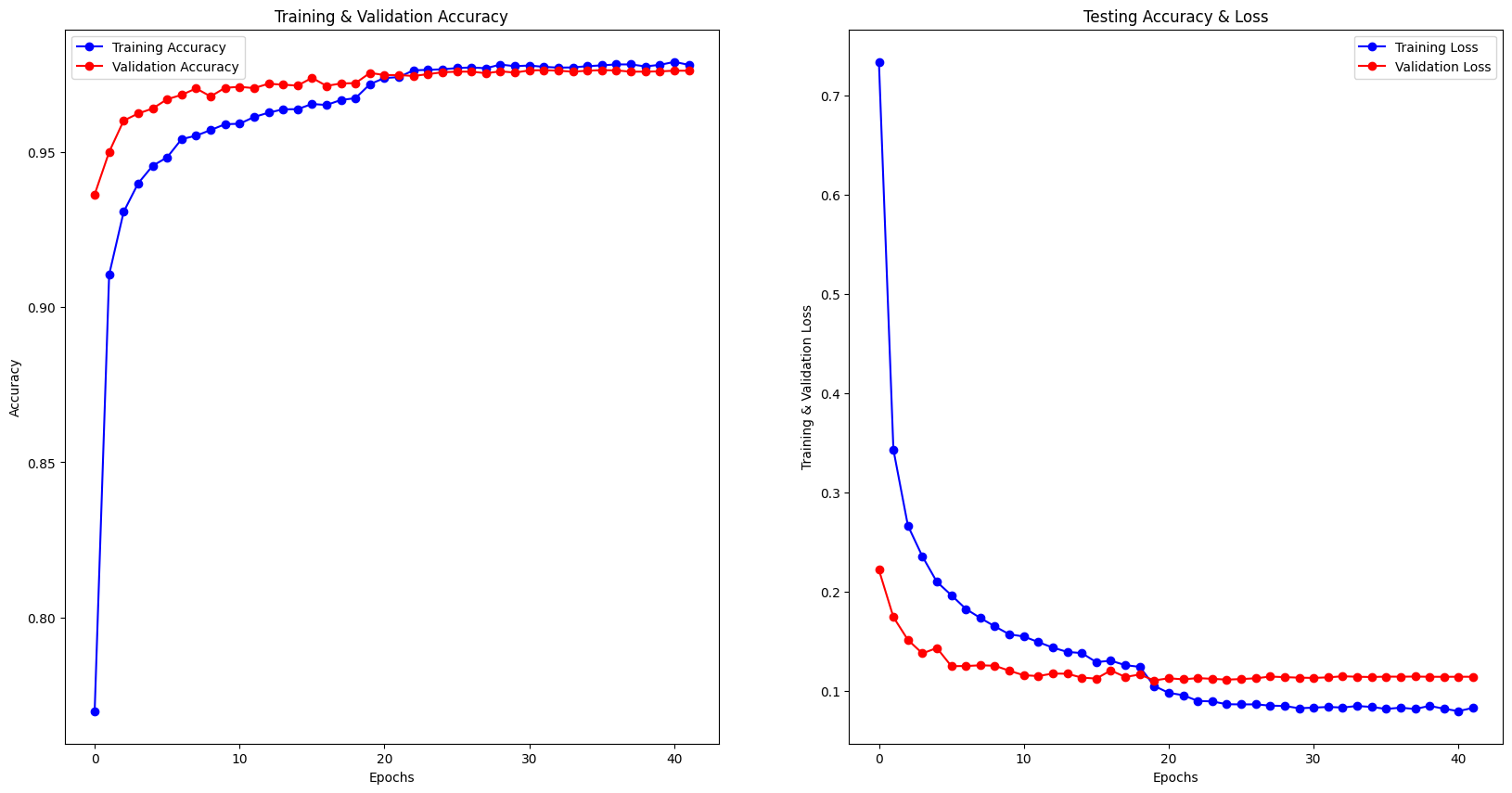
using the leaky relu activation function:



Using the ELU function :

To see which class the model is working on the best I

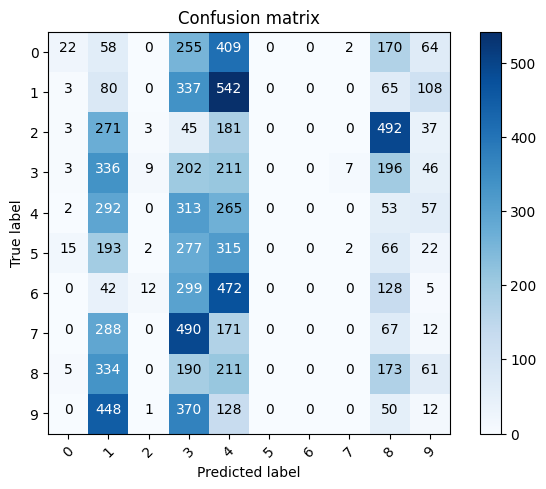
Using the relu activation function had the best result:



The accuracy reached 97 precent.

Evaluation:

To make a prediction with my classifier model I used one\_hot\_encoding on the predicted vector. And then I could give it to the confusion\_matrix function (y\_pred\_classes) which makes the confusion matrix.

After all this I used a function to plot the confusion matrix .

It seems the model had the best performance on numbers 5 , 6 and 7,which makes sense because they don’t resemble other numbers as much.