CISC 867: Project 2

Fashion-MNIST

Kariman Karm Mohamed Mousaa

I. Project Overview

Fashion-MNIST is a dataset of Zalando's article images, it contains 60,000 train set and 10,000 test set, each set have example of image, it's 28x28 grayscale images for 10 classes.

Problem Statement

We want to classification images of mnist fashion, we have 10 classes in dataset, from the available images in the sample, we want to know which of them belong to which class, we have used the data available on Kaggle.

Evaluation Metrics

To evaluate the model, we will use the accuracy matrix, it is how close the value is to its true value, it used for classification problems to be evaluated

The Formula:

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + FP + TN + FN)}$$

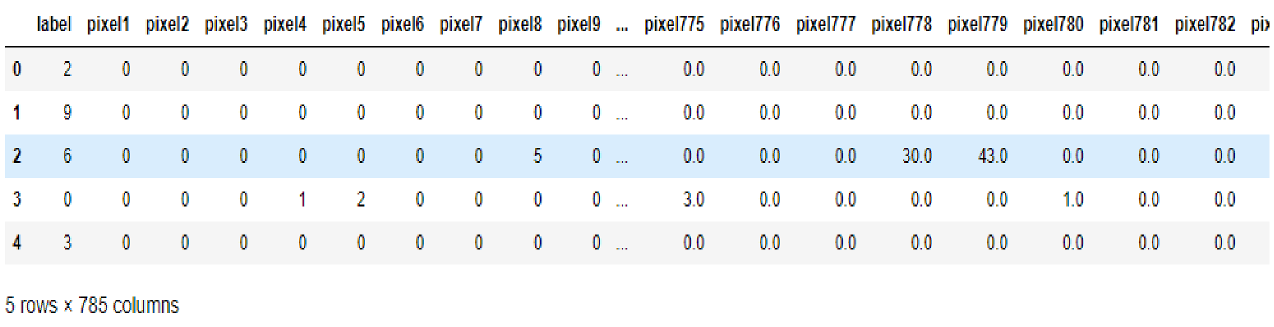
TP+TN: the number of correct predictions.

TP+FP+TN+FN: Total number of predictions.

II. Analysis

Before exploring the data, we need to import the basic libraries (NumPy, pandas, matplotlib, seaborn, sklearn, tenserflaw, keras) and the dataset.

1- - We show part of data: we found 5 rows and 785 columns as image below.



	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pix
0	2	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	9	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	6	0	0	0	0	0	0	0	5	0	...	0.0	0.0	0.0	30.0	43.0	0.0	0.0	0.0	
3	0	0	0	0	1	2	0	0	0	0	...	3.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
4	3	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

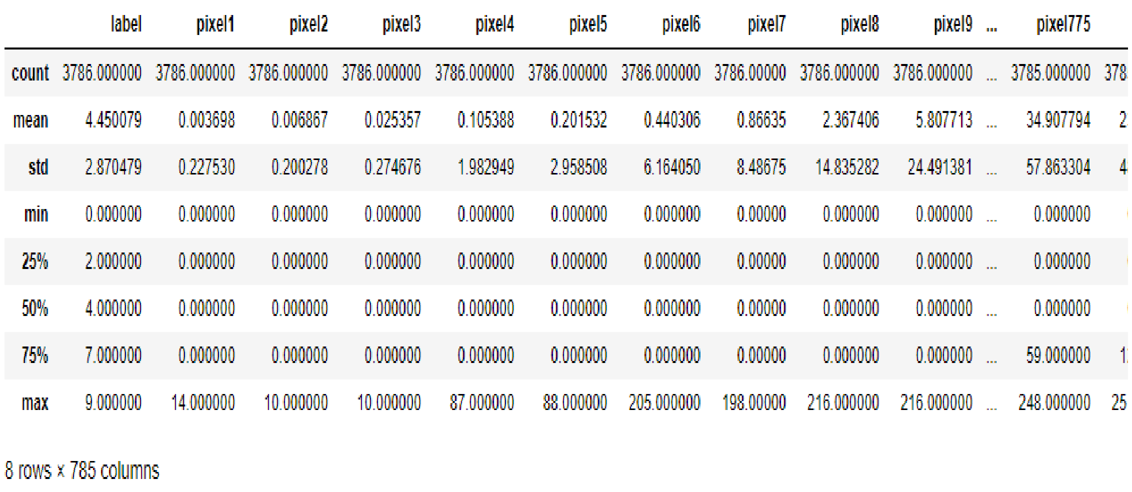
5 rows × 785 columns

train data

When describing the dataset by:

```
# Describe the data
train.describe()
```

We notice that the data is skewed to zero, but this is normal, given that the data is Grayscale:

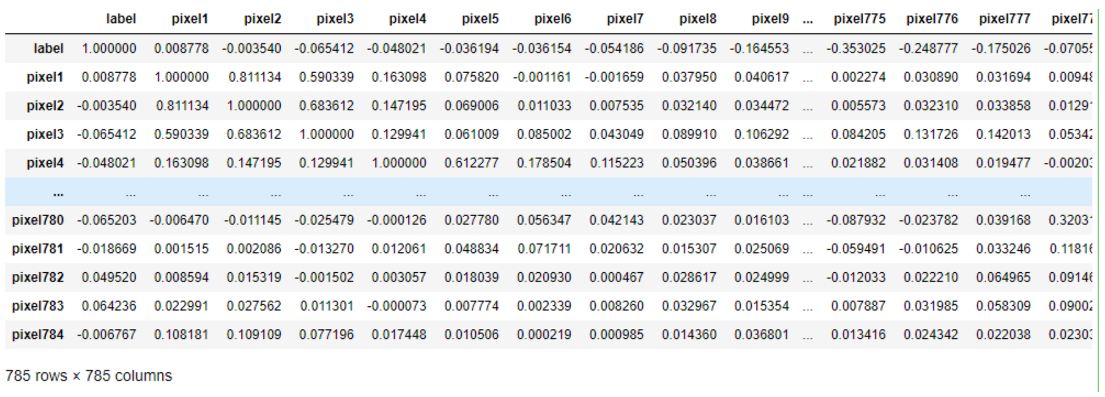


	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	
count	3786.000000	3786.000000	3786.000000	3786.000000	3786.000000	3786.000000	3786.000000	3786.00000	3786.000000	3786.000000	...	3785.000000	378
mean	4.450079	0.003698	0.006867	0.025357	0.105388	0.201532	0.440306	0.86635	2.367406	5.807713	...	34.907794	2
std	2.870479	0.227530	0.200278	0.274676	1.982949	2.958508	6.164050	8.48675	14.835282	24.491381	...	57.863304	4
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	...	0.000000	
25%	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	...	0.000000	
50%	4.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	...	0.000000	
75%	7.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	...	59.000000	1.
max	9.000000	14.000000	10.000000	10.000000	87.000000	88.000000	205.000000	198.00000	216.000000	216.000000	...	248.000000	25

8 rows × 785 columns

Describe of train

When get correlation between train data, we found a strong correlation, both positive and negative.



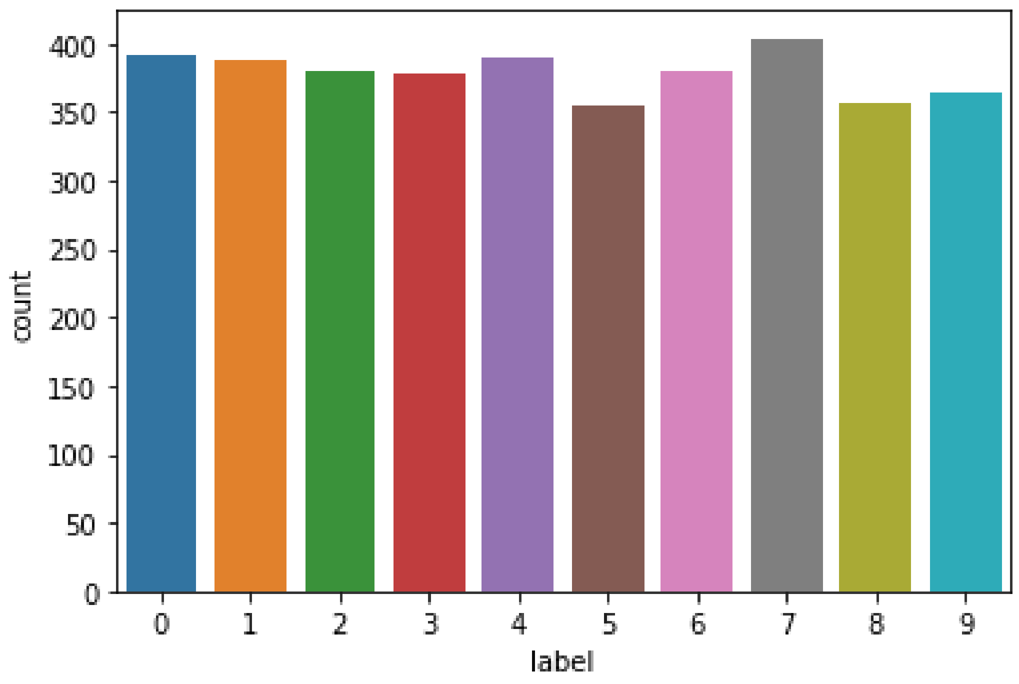
	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel77
label	1.000000	0.008778	-0.003540	-0.065412	-0.048021	-0.036194	-0.036154	-0.054186	-0.091735	-0.164553	...	-0.353025	-0.248777	-0.175026	-0.0705
pixel1	0.008778	1.000000	0.811134	0.590339	0.163098	0.075820	-0.001161	-0.001659	0.037950	0.040617	...	0.002274	0.030890	0.031694	0.0094
pixel2	-0.003540	0.811134	1.000000	0.683612	0.147195	0.069006	0.011033	0.007535	0.032140	0.034472	...	0.005573	0.032310	0.033858	0.0129
pixel3	-0.065412	0.590339	0.683612	1.000000	0.129941	0.061009	0.085002	0.043049	0.089910	0.106292	...	0.084205	0.131726	0.142013	0.0534
pixel4	-0.048021	0.163098	0.147195	0.129941	1.000000	0.612277	0.178504	0.115223	0.050396	0.038661	...	0.021882	0.031408	0.019477	-0.0020
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
pixel780	-0.065203	-0.006470	-0.011145	-0.025479	-0.000126	0.027780	0.056347	0.042143	0.023037	0.016103	...	-0.087932	-0.023782	0.039168	0.3203
pixel781	-0.018669	0.001515	0.002086	-0.013270	0.012061	0.048834	0.071711	0.020632	0.015307	0.025069	...	-0.059491	-0.010625	0.033246	0.1181
pixel782	0.049520	0.008594	0.015319	-0.001502	0.003057	0.018039	0.020930	0.000467	0.028617	0.024999	...	-0.012033	0.022210	0.064965	0.0914
pixel783	0.064236	0.022991	0.027562	0.011301	-0.000073	0.007774	0.002339	0.008260	0.032967	0.015354	...	0.007887	0.031985	0.058309	0.0900
pixel784	-0.006767	0.108181	0.109109	0.077196	0.017448	0.010506	0.000219	0.000985	0.014360	0.036801	...	0.013416	0.024342	0.022038	0.0230

785 rows × 785 columns

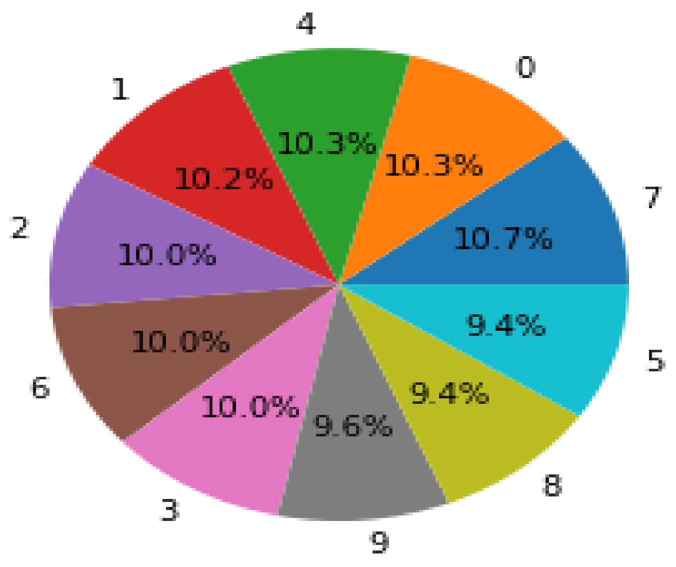
Correlation

Visualization:

When we see the distribution of the classes in label, we will find that the distribution is uniform, since the number of each class appears very close.

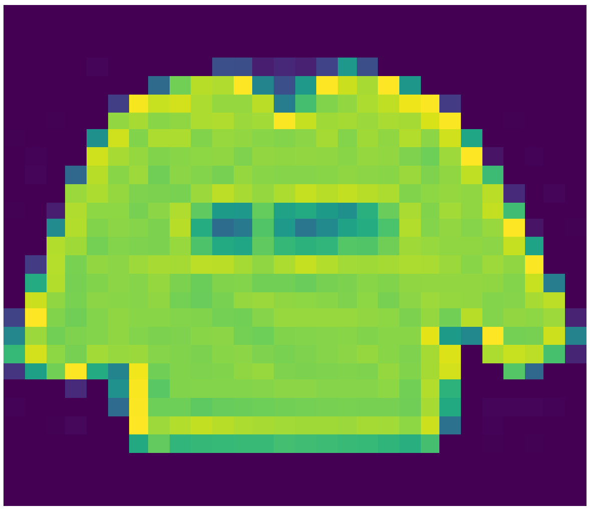


count plot

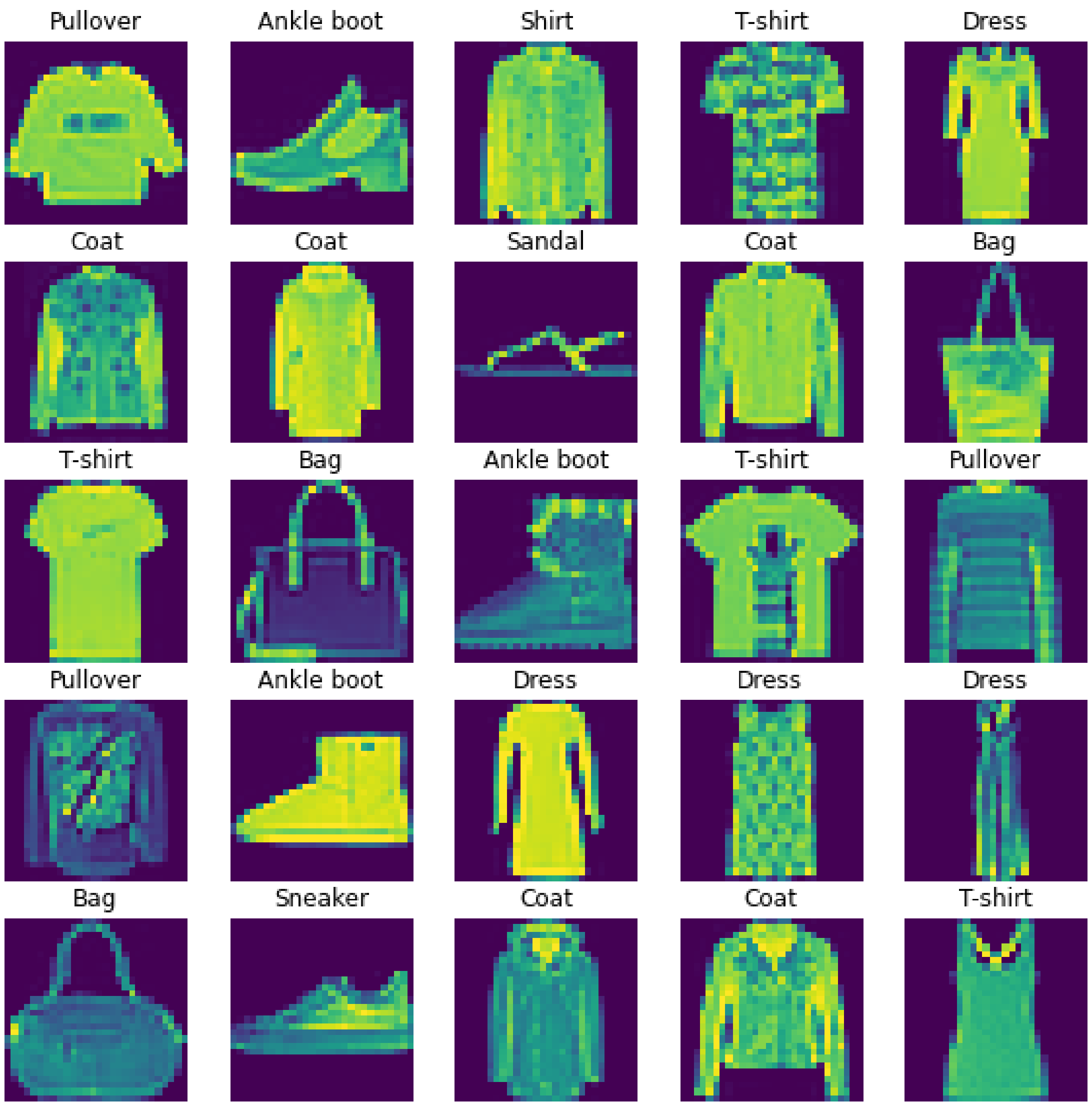


Pie chart

When we draw sample of data in train we get:



When we draw sample for each class in label for train we get:



III. Methodology

Preprocessing:

We made function for preprocessing that do:

1- transform the labels by one-hot-encoding.

2- reshape the dataset into 4D array.

3- normalize value to [0,1].

Then we uesd kfold to split data to train and validation by 5 folder.

And the shape became:

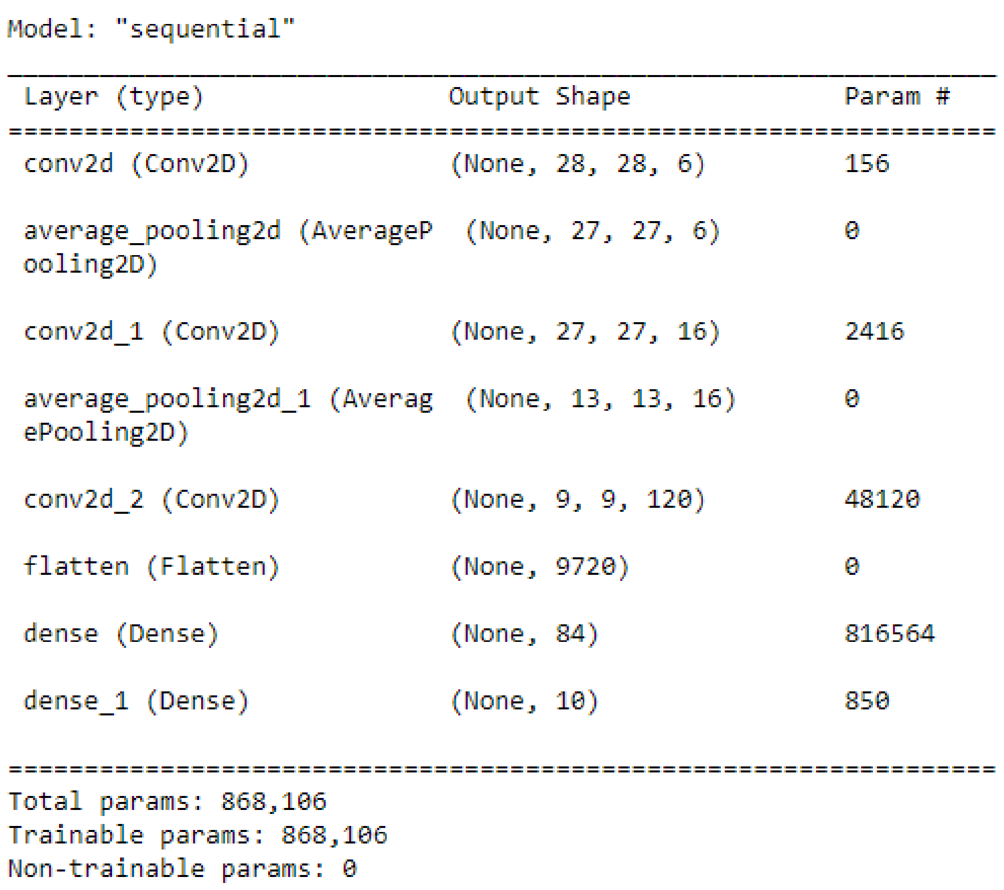
```
x_train shape:(3029, 28, 28, 1)
y_train shape:(3029, 10)
x_test shape:(757, 28, 28, 1)
y_test shape:(757, 10)
```

Models:

1- LeNet5 model:

LeNet was used in detecting handwritten cheques on MNIST dataset.
Fully connected networks and activation functions were previously known in neur al networks. LeNet-5 introduced convolutional and pooling layers. LeNet-5 is beli eved to be the base for all other ConvNets. It created by these steps:

1- instantiate an empty model.
2- c1 convolutional layer with filter=6, kernel size =5x5, strides=1x1 and tanh activation function, input shape =28x28x3 and same padding.
3- s2 average pooling layer with pooling size =5x5, strides=1x1 and valid padding.
4- c3 convolutional layer with filter=16, kernel size =5x5, strides=1x1 and tanh activation function and same padding.
5- s4 average pooling layer with pooling size =5x5, strides=1x1 and valid padding.
6- c5 full connected convolutional layer with filter=120, kernel size =5x5, strides=1x1 and tanh activation function and same padding.
7- flatten the cnn output so that we can connect it with full connected layers.
8- Fc6 full connected layer with 84 neuron and tanh activation function.
9- output layer with softmax activation.
10- optimizer Adam with learning rate=.01 in second trial .001 and metrics=accuracy.
11- fit model with batch size=250 and epochs=50 then in trial2 batch size=200, epochs=20.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 6)         156

 average_pooling2d (AverageP  (None, 27, 27, 6)        0
 ooling2D)

 conv2d_1 (Conv2D)           (None, 27, 27, 16)        2416

 average_pooling2d_1 (Averag  (None, 13, 13, 16)       0
 ePooling2D)

 conv2d_2 (Conv2D)           (None, 9, 9, 120)         48120

 flatten (Flatten)           (None, 9720)              0

 dense (Dense)               (None, 84)                816564

 dense_1 (Dense)             (None, 10)                850

=================================================================
Total params: 868,106
Trainable params: 868,106
Non-trainable params: 0
```

summary

CNN model

CNN is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various objects in the image.

We created 7 layers in the first layer we used Conv2D with 32 filter kernel size=3x3, activation=relu, input shape=28x28x3 then Normalization. In layer 2,3,4 we used Conv2D + Maxpooling +Normalization+ Dropout. Then Flatten, in 5,6 layer we used

Dense+ Normalization + Dropout with activation='relu'. In output layer we used activation=softmax.

VGG16 model

1- X forms the training images, and y forms the training labels.

2- Converting Labels to one hot encoded format.

3- Convert the images into 3 channels.

4- Reshape images as per the tensor format required by tensorflow.

5- Resize the images 48*48 as required by VGG16.

6- Normalize the data and change data type.

7- Splitting train data as train and validation data.

8- Preprocessing the input.

9- Create base model of VGG16 with weights='ImageNet', include top=False, input shape = (150, 150, 3) and classes = 10.

10- Extracting features

11-Saving the features so that they can be used for future.

12- Flatten extracted features.

13- Add Dense and Dropout layers on top of VGG16 pre-trained.

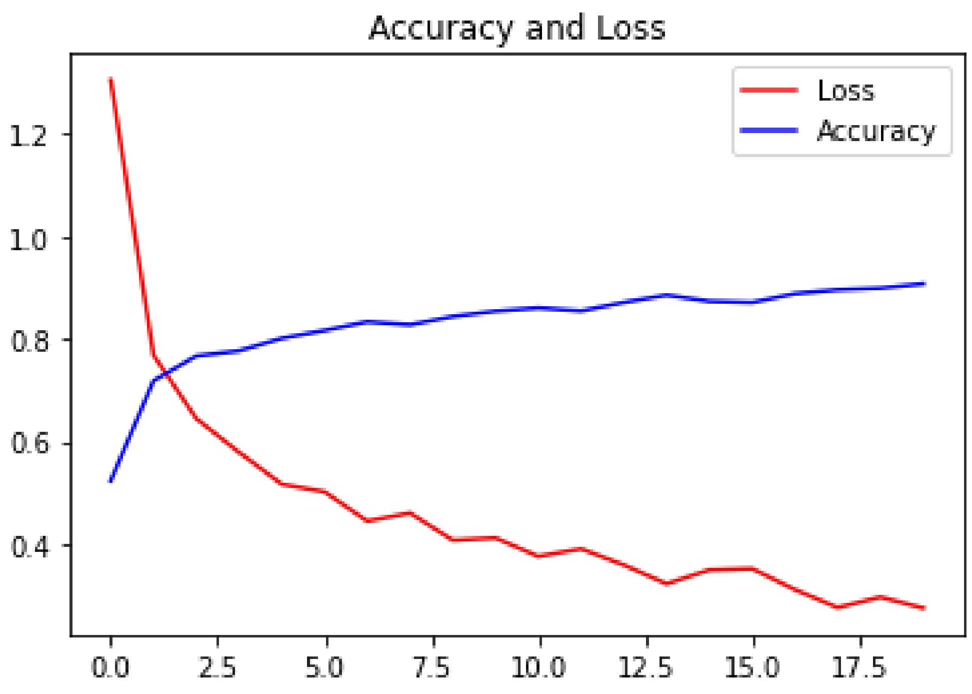
14- fit the model   batch size=256

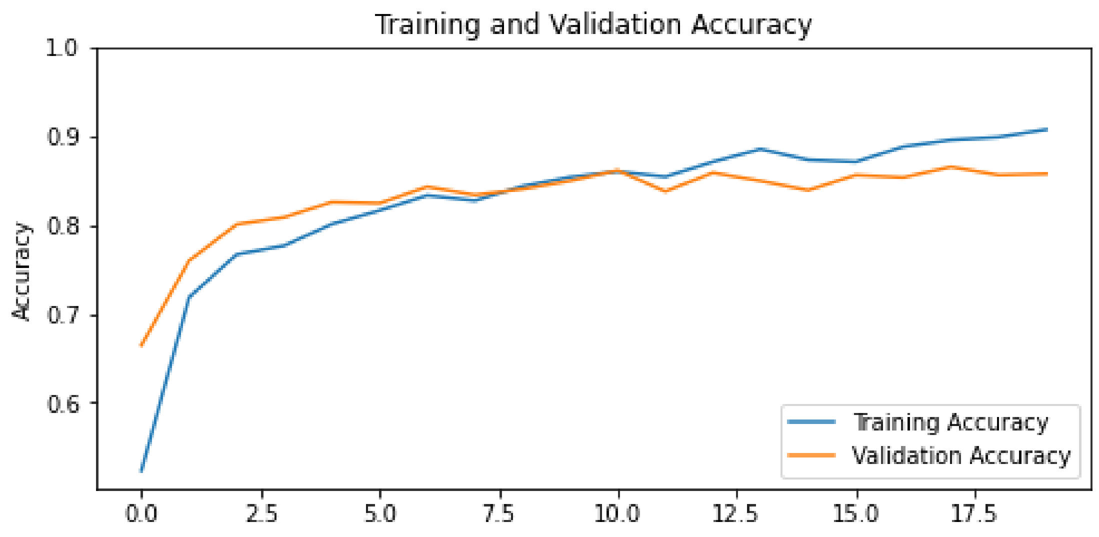
And epochs=50 verbose=1.

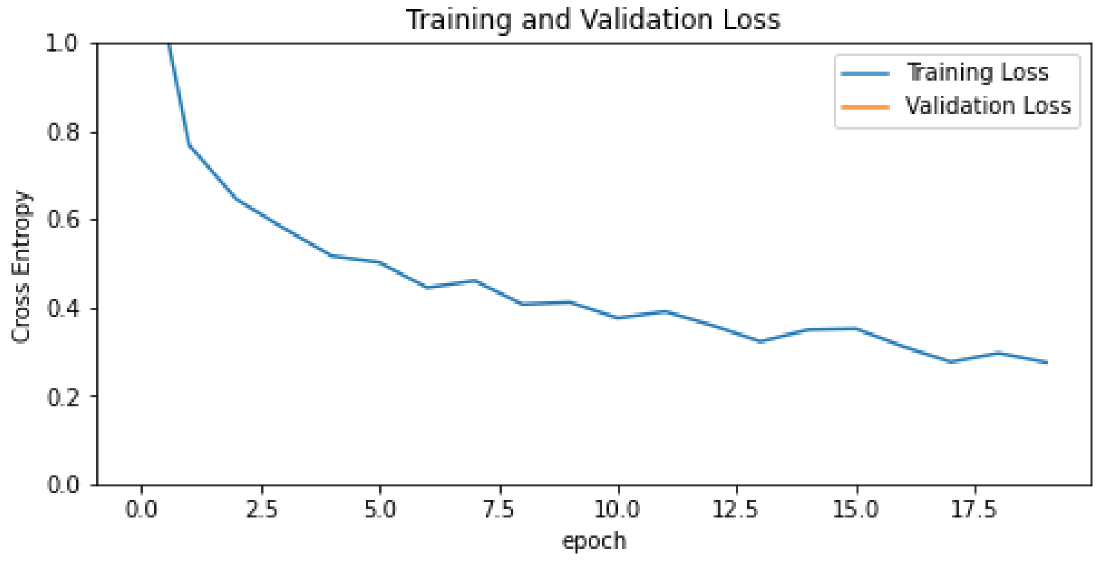
IV. Results

Model Evaluation

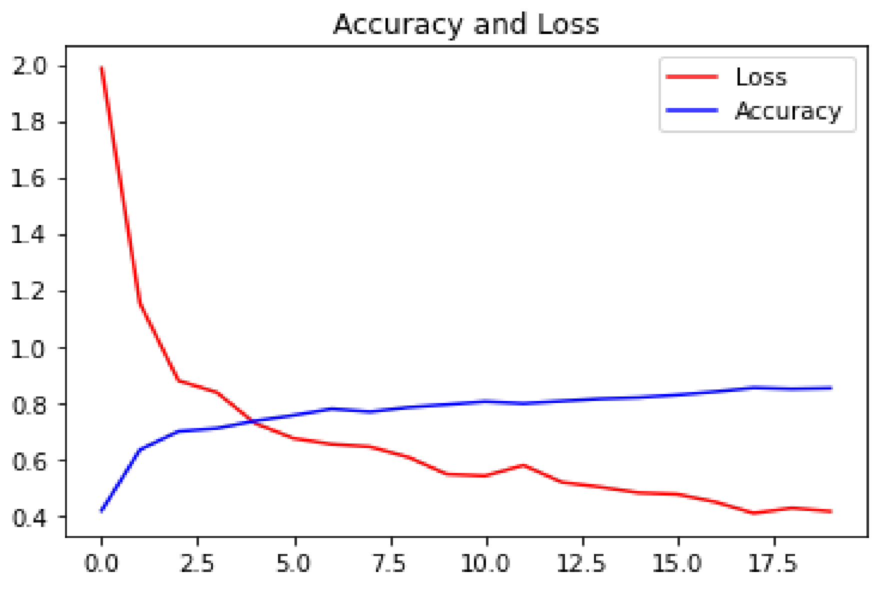
1-Benchmark Model: LeNet5 model, it was the best model and we could have improv ed the performance more, but because of the weak internet and also a device malfun ction, the output was swinging between 85% and 95%.





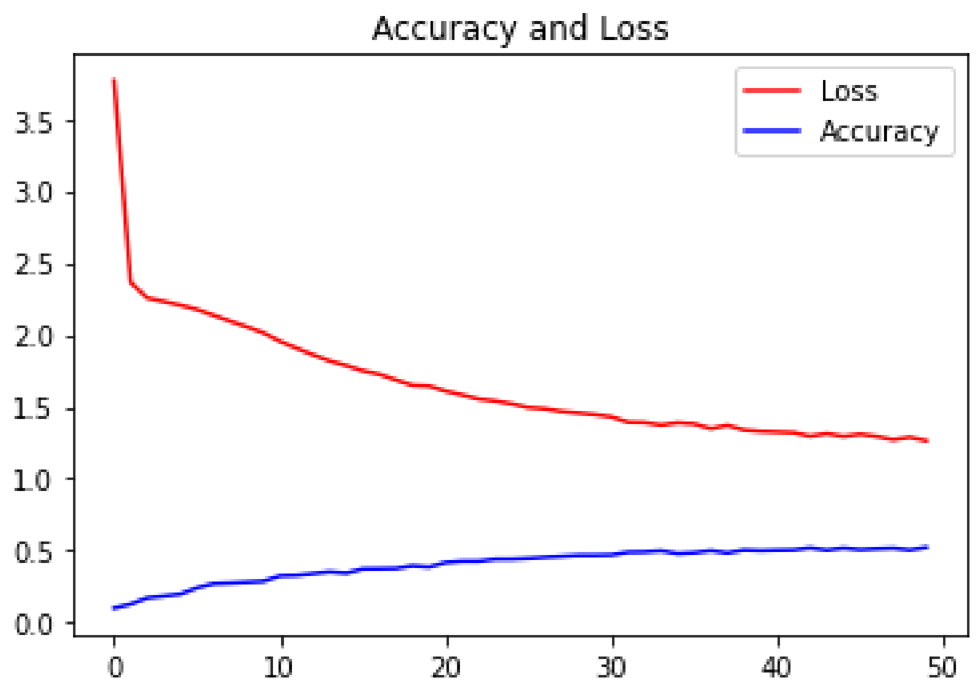


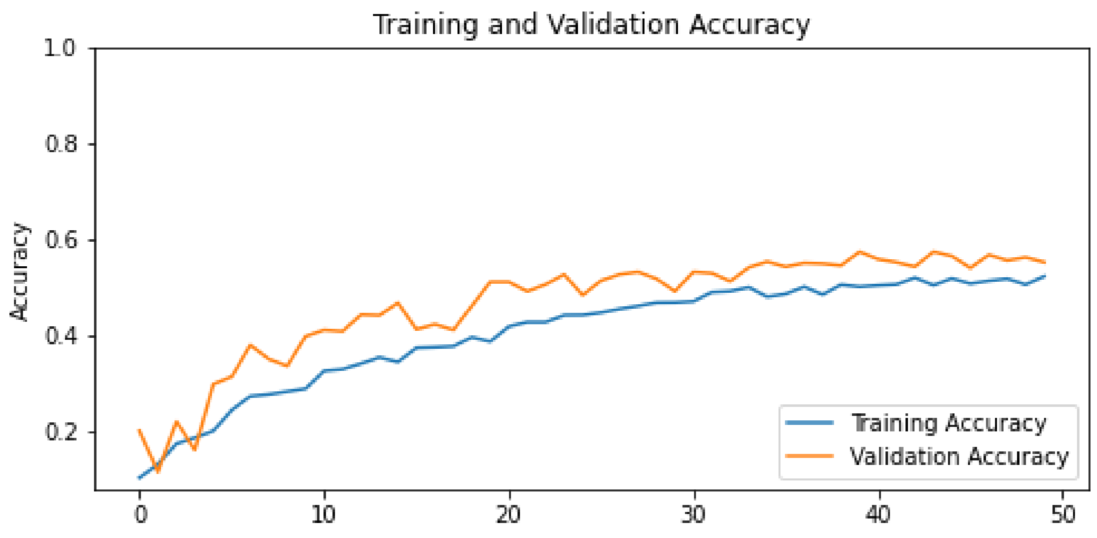
2- CNN model It was the worst in performance Accuracy: %26.02.



2- VGG16 model It can be said that it is average in performance, but lenet5 is the best.

```
Accuracy: %55.15
```





References:

1- https://www.kaggle.com/zalando-research/fashionmnist

2- http://yann.lecun.com/exdb/lenet

3- https://analyticsindiamag.com/complete-tutorial-on-lenet-5-guide-to-/begin-with-cnns