# ASSIGNMENT 3

## CSE 574 Intro to Machine Learning (Section D)


### Team 113

### Dhruv Patel - 50432968

### Pratik Malani- 50416266


We certify that the code and data in this assignment were generated independently, using only the tools and resources defined in the course and that I did not receive any external help, coaching or contributions during the production of this work.

**Part – 1 Building a Basic Neural Network**

1. The dataset is about income of dataset of people. The dataset contains thirteen column attributes with 32561 entries in the rows. It gives out details of different income class and the information with respect to the income for us to train the neural network model. It consists of int, object and float

| | age | workclass | fnlwgt | education | education.num | marital.status | occupation | relationship | capital.gain | capital.loss | hours.per.week | native.country | income |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 90 | ? | 77053.0 | HS-grad | 9 | Widowed | ? | Not-in-family | 0 | 4356 | 40 | United-States | <=50K |
| 1 | 82 | Private | 186061.0 | HS-grad | 9 | Widowed | Exec-managerial | Not-in-family | 0 | 4356 | 18 | United-States | <=50K |
| 2 | 66 | ? | NaN | Some-college | 10 | Widowed | ? | Unmarried | 0 | 4356 | 40 | United-States | <=50K |
| 3 | 54 | Private | 140359.0 | 7th-8th | 4 | Divorced | Machine-op-inspct | Unmarried | 0 | 3900 | 40 | United-States | <=50K |
| 4 | 41 | Private | 264663.0 | Some-college | 10 | Separated | Prof-specialty | Own-child | 0 | 3900 | 40 | United-States | <=50K |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 32556 | 22 | Private | 310152.0 | Some-college | 10 | Never-married | Protective-serv | Not-in-family | 0 | 0 | 40 | United-States | <=50K |
| 32557 | 27 | Private | 257302.0 | Assoc-acdm | 12 | Married-civ-spouse | Tech-support | Wife | 0 | 0 | 38 | United-States | <=50K |
| 32558 | 40 | Private | 154374.0 | HS-grad | 9 | Married-civ-spouse | Machine-op-inspct | Husband | 0 | 0 | 40 | United-States | >50K |
| 32559 | 58 | Private | 151910.0 | HS-grad | 9 | Widowed | Adm-clerical | Unmarried | 0 | 0 | 40 | United-States | <=50K |
| 32560 | 22 | Private | 201490.0 | HS-grad | 9 | Never-married | Adm-clerical | Own-child | 0 | 0 | 20 | United-States | <=50K |

32561 rows × 13 columns

| | age | fnlwgt | education.num | capital.gain | capital.loss | hours.per.week |
|---|---|---|---|---|---|---|
| count | 32561.000000 | 3.256000e+04 | 32561.000000 | 32561.000000 | 32561.000000 | 32561.000000 |
| mean | 38.581647 | 1.897801e+05 | 10.080679 | 1077.648844 | 87.303830 | 40.437456 |
| std | 13.640433 | 1.055511e+05 | 2.572720 | 7385.292085 | 402.960219 | 12.347429 |
| min | 17.000000 | 1.228500e+04 | 1.000000 | 0.000000 | 0.000000 | 1.000000 |
| 25% | 28.000000 | 1.178242e+05 | 9.000000 | 0.000000 | 0.000000 | 40.000000 |
| 50% | 37.000000 | 1.783630e+05 | 10.000000 | 0.000000 | 0.000000 | 40.000000 |
| 75% | 48.000000 | 2.370545e+05 | 12.000000 | 0.000000 | 0.000000 | 45.000000 |
| max | 90.000000 | 1.484705e+06 | 16.000000 | 99999.000000 | 4356.000000 | 99.000000 |

datatypes for the entries. Main statistics of the dataset is shown below:

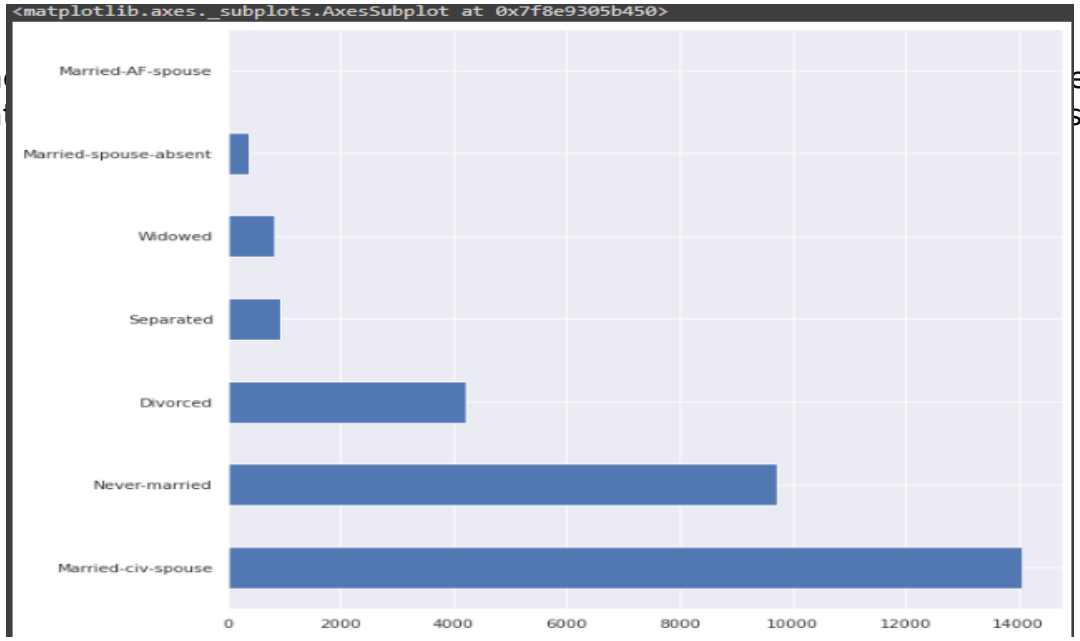
```
age                0.293670
fnlwgt             0.119670
capital.gain       0.010920
workclass          2.199324
education         10.333764
education.num      0.608087
marital.status     2.580134
occupation         5.959850
relationship       1.418341
capital.loss       0.020288
hours.per.week     0.407462
native.country    36.382567
income             0.248922
dtype: float64
age                0.032374
fnlwgt             0.005159
capital.gain       0.005486
workclass          0.909972
education         14.533570
education.num      0.028900
marital.status     2.244051
occupation        16.237405
relationship       2.564284
capital.loss       0.008614
hours.per.week     0.014944
native.country    37.275573
income             0.186966
dtype: float64
```

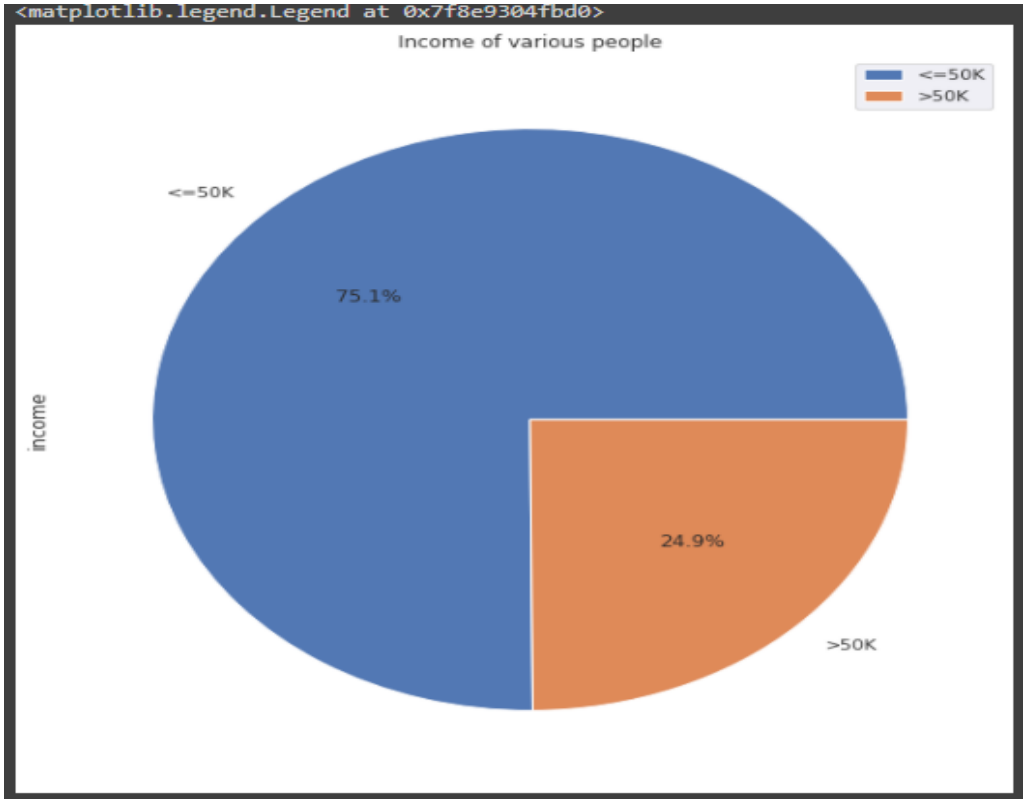
```
age               int64
workclass        object
fnlwgt          float64
education        object
education.num     int64
marital.status   object
occupation       object
relationship     object
capital.gain      int64
capital.loss      int64
hours.per.week    int64
native.country   object
income           object
dtype: object
```

Mean and variance of dataset along with the datatypes.

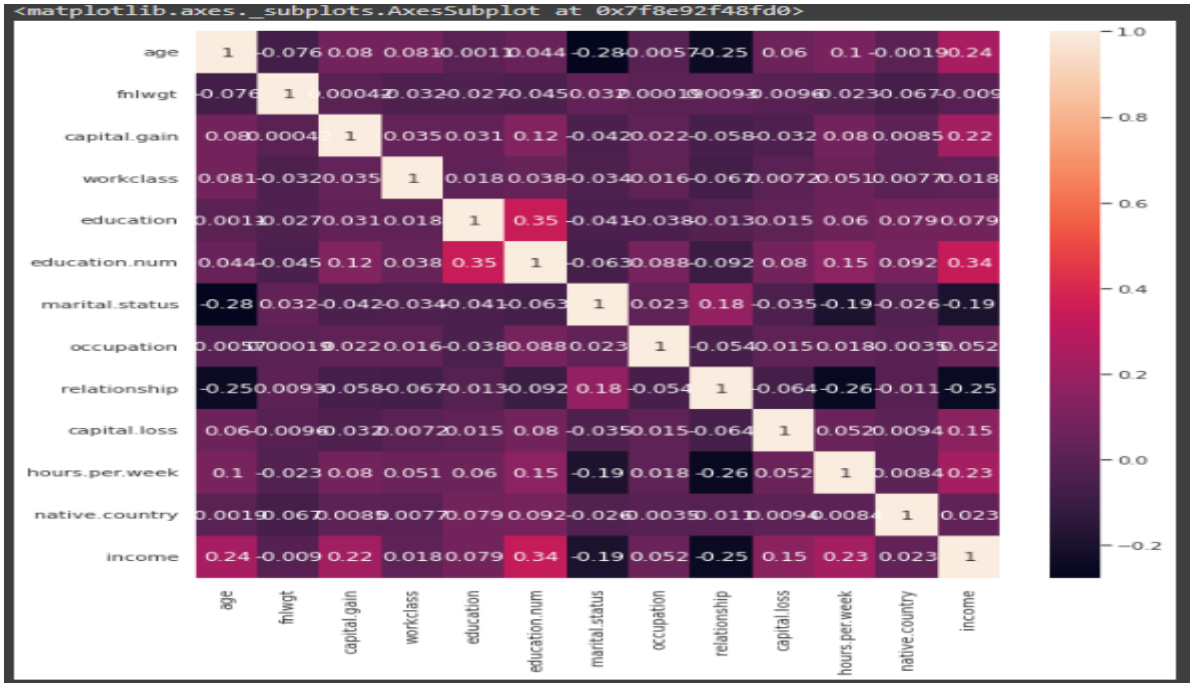
2. Visualization graphs are shown below:

Th e
da se".






The pie charts above show the distribution of income in the dataset. Here it is seen that about 75% of the people have income less than or equal to o 50K and about 25% of the people have income more than 50K.

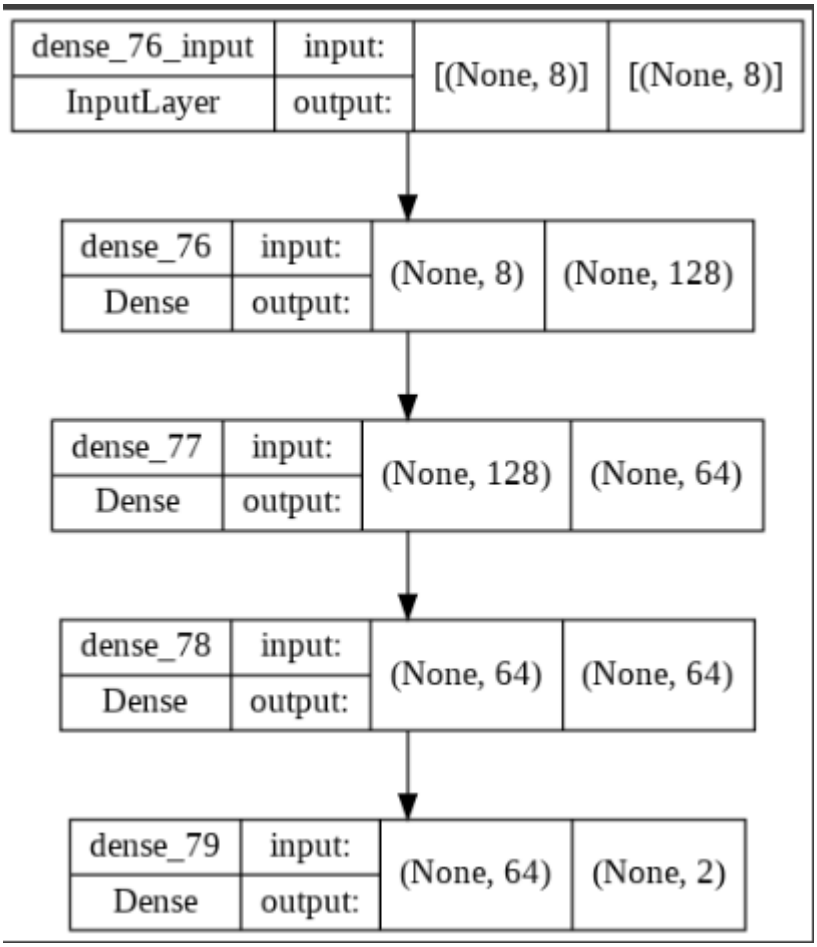


The graph above shows the correlation heatmap of the dataset. This was used to see which columns have a correlation with our target column, which is the income. 'Education.num' has the highest correlation.

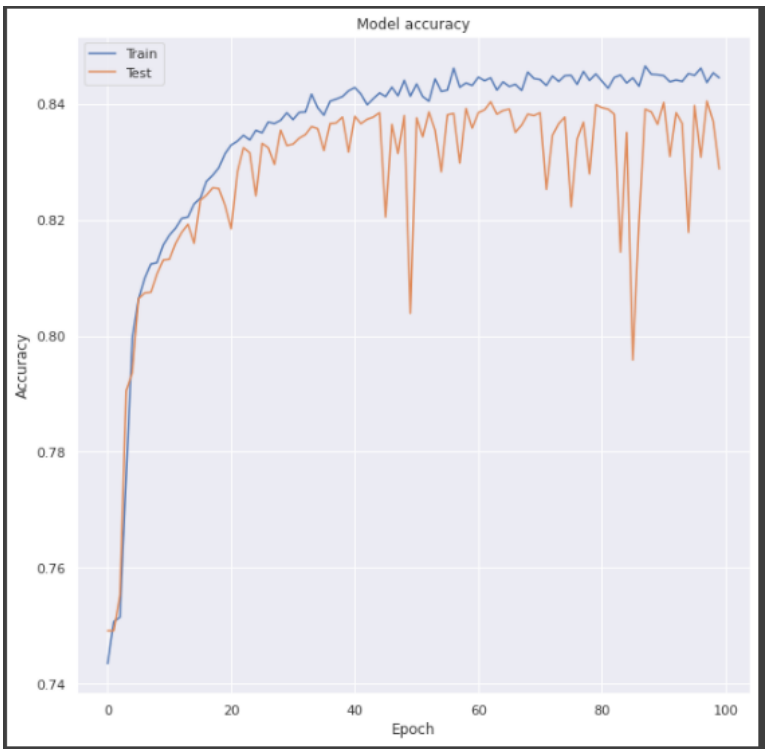
3. For preprocessing of the dataset, we first dropped the null and "?" values from the dataset to get a complete dataset. We then normalized all the values in the dataset and assigned the similar string values of a column in the data set with a unique integer using ".cat.codes" function. We then scaled and divided the dataset into train (80%) and test (20%) data using sklearns. All this contributes to increase the accuracy of the neural network model.
   The heat map shown above also shows the correlation of different columns in the dataset which helps us to make a more knowledgeable choice of which columns to include in our model to help us get a more accurate final model.

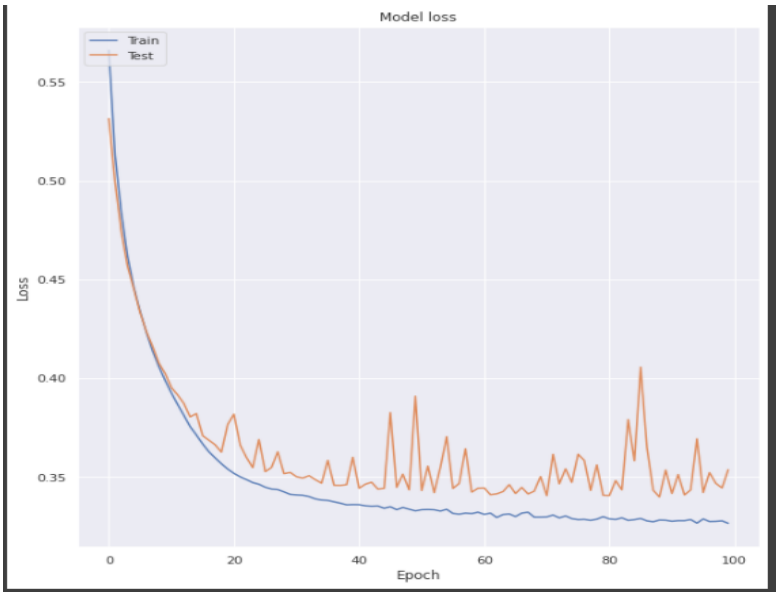
4. The neural network architecture is shown below:

   We have the architecture as shown below with an input layer, three hidden layers and an output layer. We have 8 in put variables in the input layer, 128, 64 and 64 for the hidden layers respectively and two for the output layer




5. The two graphs that depict the comparison between test and training accuracy and test
   and training loss are shows below respectively:




```
505/505 [==============================] - 1s 1ms/step - loss: 0.3346 - accuracy: 0.8411
The accuracy of the Training data set is  84.11046862602234
249/249 [==============================] - 0s 1ms/step - loss: 0.3537 - accuracy: 0.8288
The accuracy of the Test data set is  82.87653923034668
```

Below is the model summery-

```
Model: "sequential_19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_76 (Dense)            (None, 128)               1152

 dense_77 (Dense)            (None, 64)                8256

 dense_78 (Dense)            (None, 64)                4160

 dense_79 (Dense)            (None, 2)                 130

=================================================================
Total params: 13,698
Trainable params: 13,698
Non-trainable params: 0
_________________________________________________________________
```

Part – 2 Optimizing the Neural Network

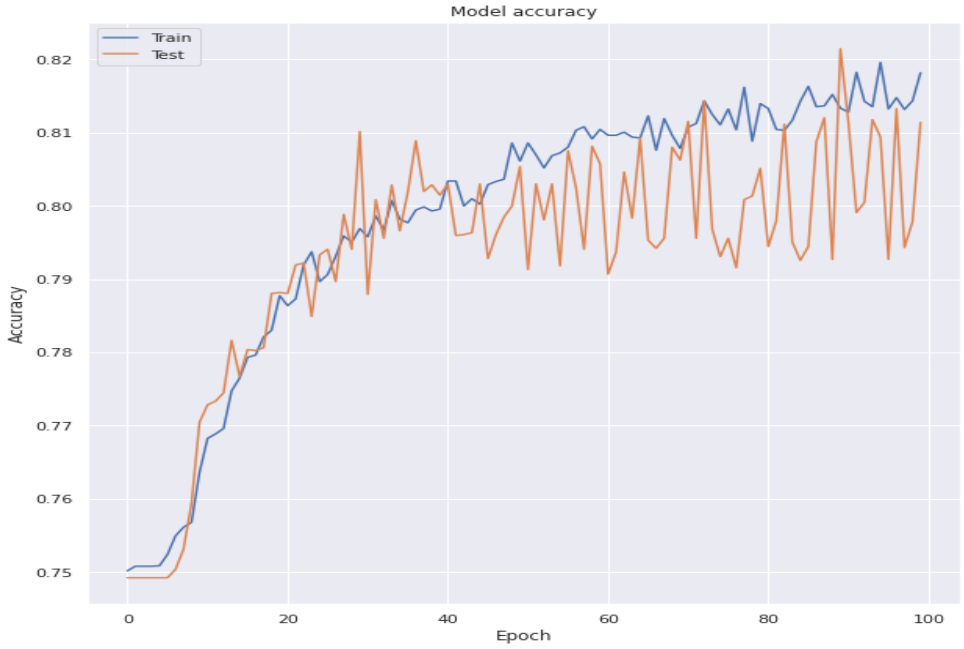
1. Below we have provided the tables along with the graph for different NN setups:
   Below is the table with different dropout values:
   For the first part we used different dropout values which were 10, 20 and 4%. on updating our model with the following dropout values, we found different results. Initially when we increased the dropout to 20%, we found that the accuracy of dropped. For this reason, we tried to take a smaller dropout value for the next and dropped it down to 4%. We found that we get the best accuracy at 4% dropout value, so we keep the model to work at that value.

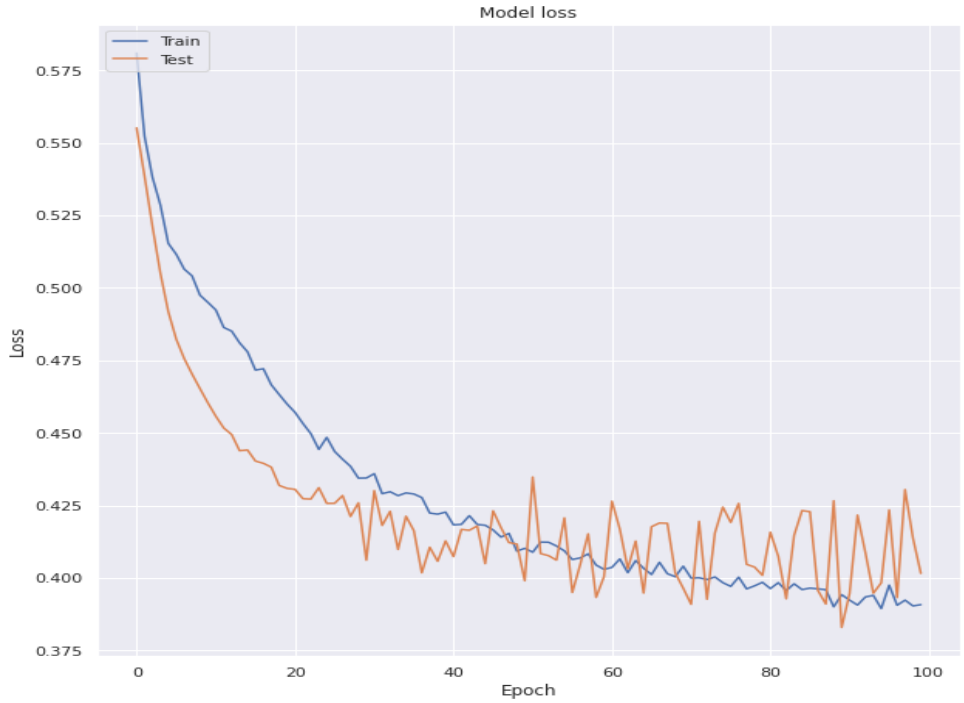
| | Setup-1 | Accuracy | Setup-2 | Accuracy | Setup-3 | Accuracy |
|---|---|---|---|---|---|---|
| Dropout | 10% | 83.78% | 20% | 83.93% | 1.5% | 83.810% |
| Optimizer | SGD | | SGD | | SGD | |
| Activation functions | RELU<br>RELU<br>RELU<br>Softmax | | RELU<br>RELU<br>RELU<br>Softmax | | RELU<br>RELU<br>RELU<br>Softmax | |


Graphs for Setup-1

Model accuracy for 10% dropout-

Graphs for Setup-2

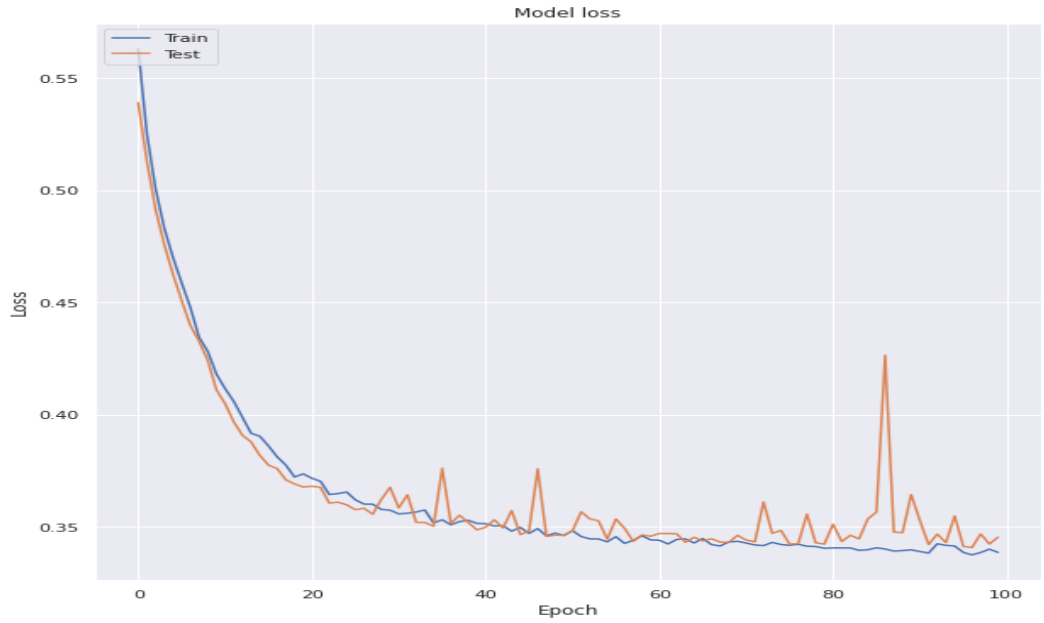



Model accuracy for 20% dropout is –

```
505/505 [==============================] - 1s 3ms/step - loss: 0.3286 - accuracy: 0.8442
The accuracy of the Training data set is  84.4200849533081
249/249 [==============================] - 1s 2ms/step - loss: 0.3443 - accuracy: 0.8393
The accuracy of the Test data set is  83.93261432647705
```

Graphs for Setup-3


Model accuracy for 4% dropout is –

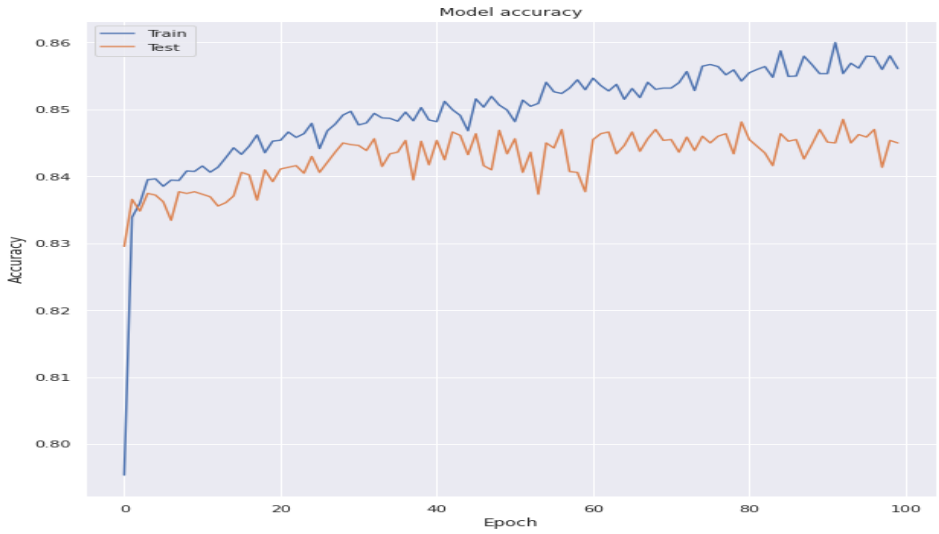
```
505/505 [==============================] - 1s 1ms/step - loss: 0.3253 - accuracy: 0.8466
The accuracy of the Training data set is  84.66159105300903
249/249 [==============================] - 0s 1ms/step - loss: 0.3421 - accuracy: 0.8382
The accuracy of the Test data set is  83.81946086883545
```

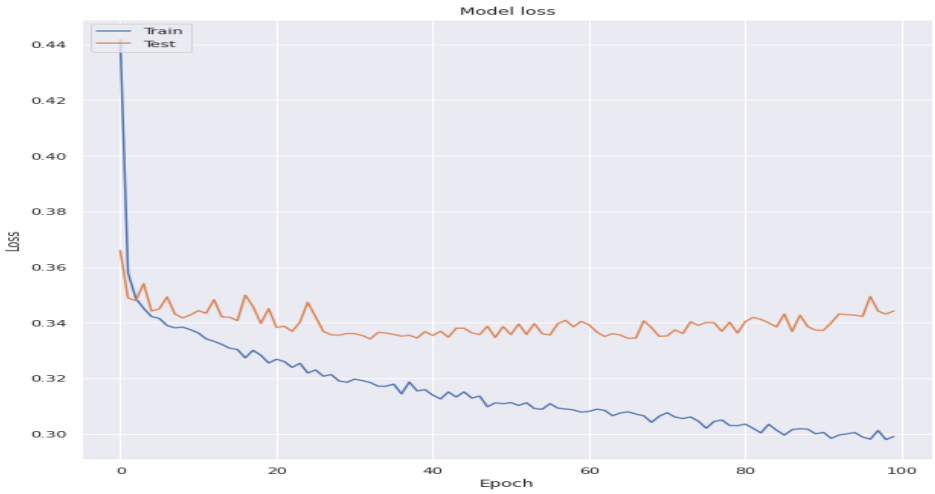
We then took the next hyperparameter which is optimizers and tried with different optimizers to find the best accuracy we can find. For this we started with ADAM optimizer. We then moved on to try RMSprop optimizer and finally we tried using Adagrad optimizer to see which one gives us the best accuracy possible. On final comparison we found that RMSprop optimizer gives the best accuracy for our model and that is why we kept it as the final optimizer.

Below is the table with different Optimizer values:

| | Setup-1 | Accuracy | Setup-2 | Accuracy | Setup-3 | Accuracy |
|---|---|---|---|---|---|---|
| Dropout | 1.5% | 84.00% | 1.5% | 84.38% | 1.5% | 82.33% |
| Optimizer | ADAM | | RMSprop | | Adagrad | |
| Activation functions | RELU<br>RELU<br>RELU<br>Softmax | | RELU<br>RELU<br>RELU<br>Softmax | | RELU<br>RELU<br>RELU<br>Softmax | |

Graphs for Setup-1

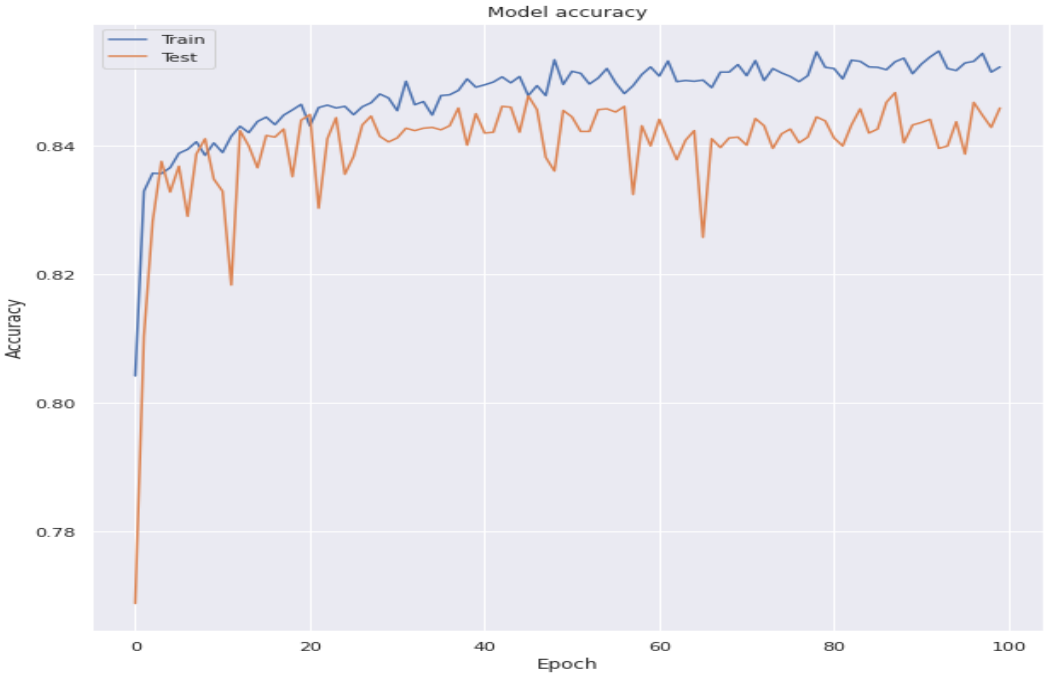





Model accuracy for ADAM optimizer-

```
505/505 [==============================] - 1s 1ms/step - loss: 0.2830 - accuracy: 0.8661
The accuracy of the Training data set is  86.60598397254944
249/249 [==============================] - 0s 2ms/step - loss: 0.3499 - accuracy: 0.8401
The accuracy of the Test data set is  84.00804400444031
```

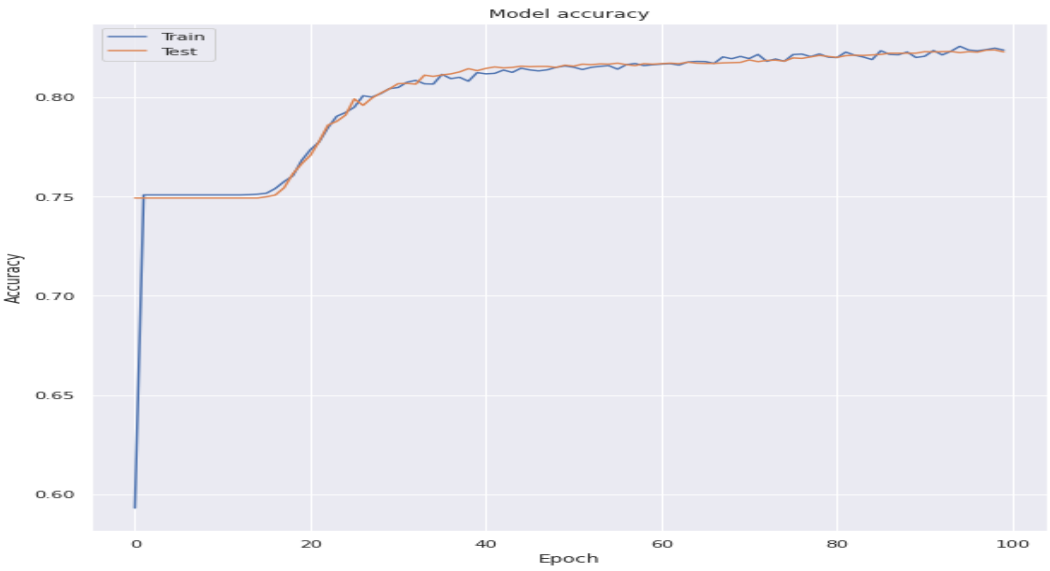
Graphs for Setup-2

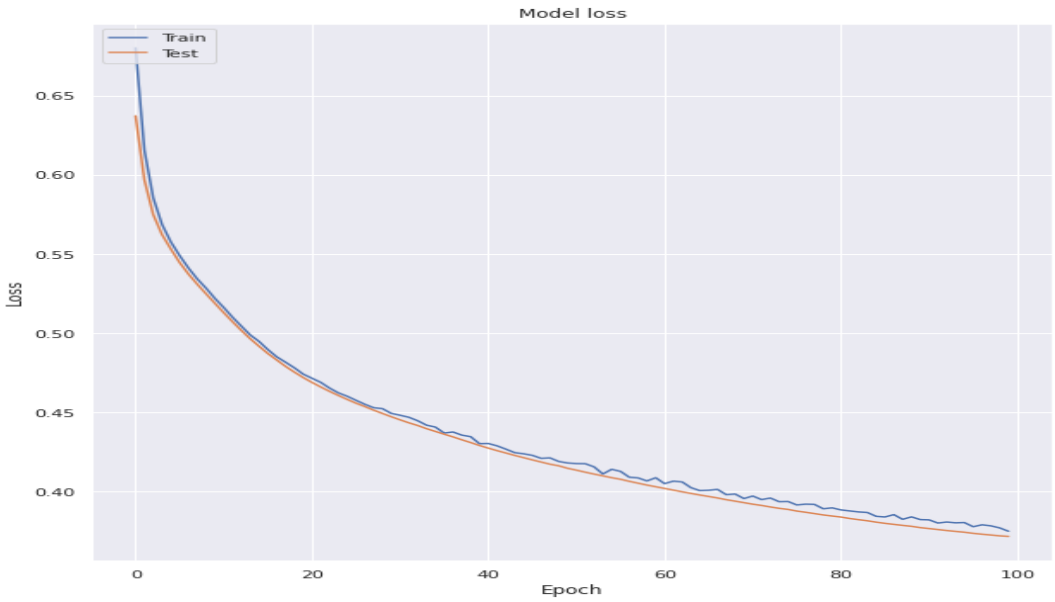



Model accuracy for RMSprop optimizer-

```
505/505 [==============================] - 1s 1ms/step - loss: 0.3098 - accuracy: 0.8547
The accuracy of the Training data set is  85.46659350395203
249/249 [==============================] - 0s 1ms/step - loss: 0.3417 - accuracy: 0.8439
The accuracy of the Test data set is  84.3852162361145
```

Graphs for Setup-3






Model accuracy for Adagrad optimizer-

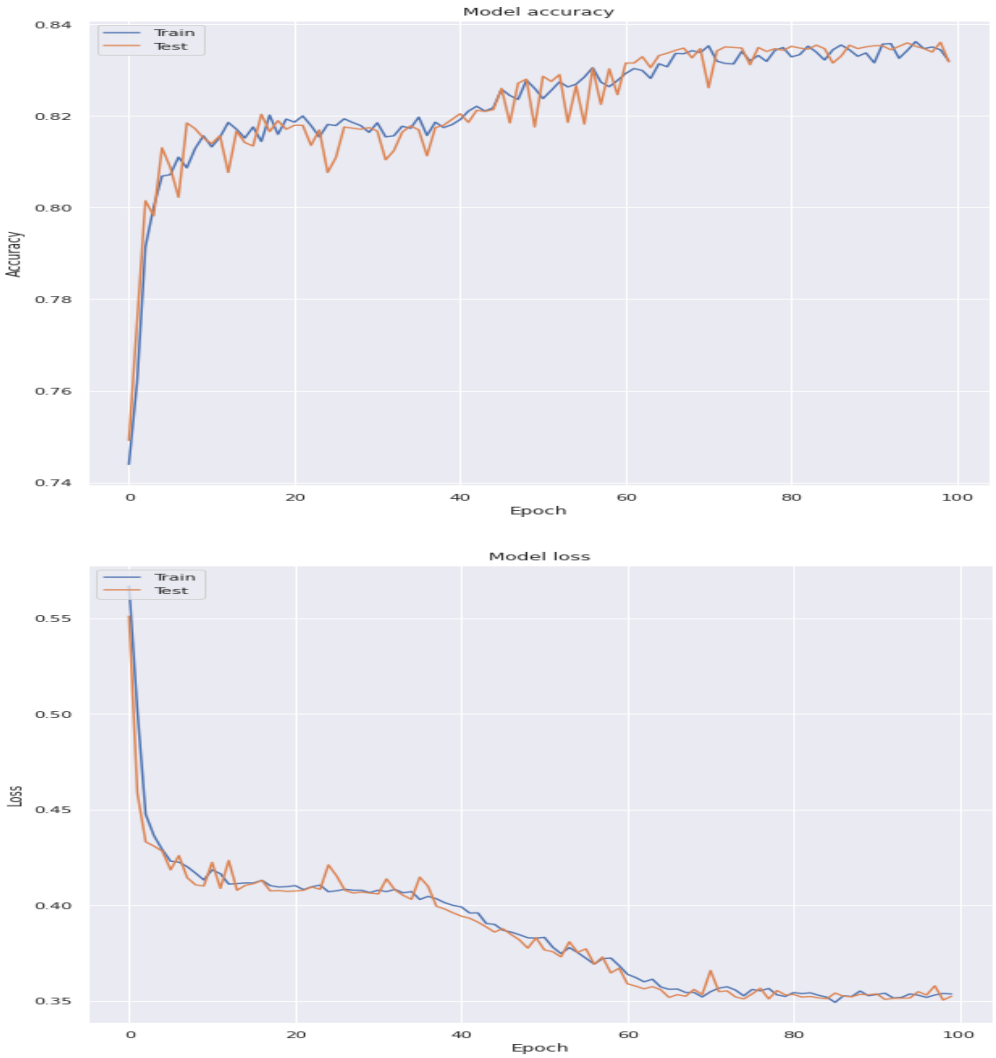
```
505/505 [==============================] - 1s 1ms/step - loss: 0.3675 - accuracy: 0.8265
The accuracy of the Training data set is  82.64908194541931
249/249 [==============================] - 0s 1ms/step - loss: 0.3738 - accuracy: 0.8234
The accuracy of the Test data set is  82.33593106269836
```

We then moved on to the next hyperparameter which is activation functions. Below is the table which lists all the combination of activation functions we used for different setups. On comparison we find that softplus activation function with the combination of RELU works the best and gives the best accuracy for our model.

Below is the table with different Activation functions:

| | Setup-1 | Accuracy | Setup-2 | Accuracy | Setup-3 | Accuracy |
|---|---|---|---|---|---|---|
| Dropout | 4% | 84.05% | 4% | 84.22 % | 4% | 83.94% |
| Optimizer | RMSprop | | RMSprop | | RMSprop | |
| Activation functions | RELU<br>RELU<br>RELU<br>Softmax | | RELU<br>RELU<br>RELU<br>Softplus | | RELU<br>RELU<br>RELU<br>Softsign | |

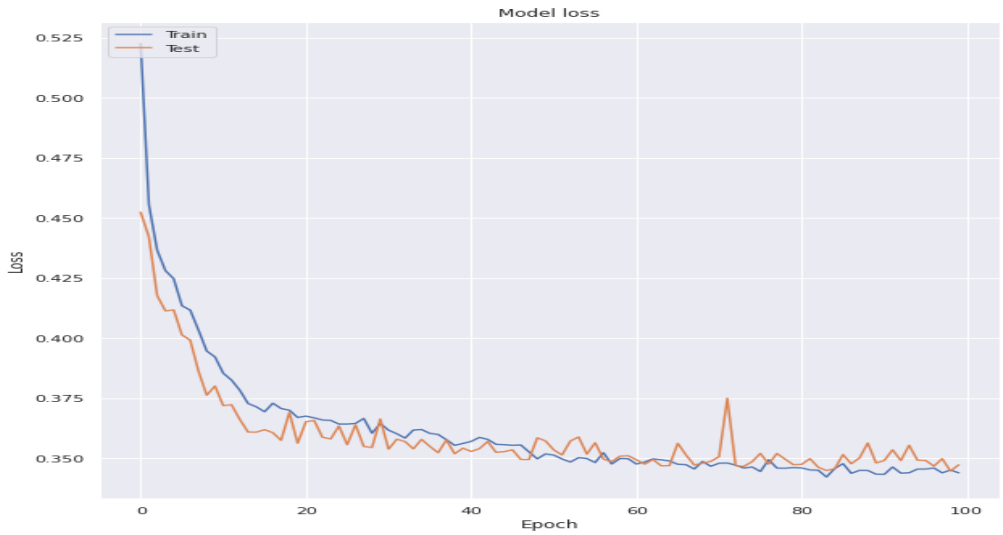
Graphs for Setup-1




Model accuracy for Setup 1

```
505/505 [==============================] - 1s 1ms/step - loss: 0.3208 - accuracy: 0.8501
The accuracy of the Training data set is  85.01455187797546
249/249 [==============================] - 0s 1ms/step - loss: 0.3630 - accuracy: 0.8406
The accuracy of the Test data set is  84.05833840370178
```

Graphs for Setup-2


Model accuracy for Setup 2

```
505/505 [==============================] - 1s 1ms/step - loss: 0.3089 - accuracy: 0.8544
The accuracy of the Training data set is  85.44182181358337
249/249 [==============================] - 0s 1ms/step - loss: 0.3480 - accuracy: 0.8422
The accuracy of the Test data set is  84.2217743396759
```

Graphs for Setup-3


Model accuracy for Setup 3

```
505/505 [==============================] - 1s 1ms/step - loss: 0.3612 - accuracy: 0.8497
The accuracy of the Training data set is  84.96501445770264
249/249 [==============================] - 0s 1ms/step - loss: 0.3762 - accuracy: 0.8395
The accuracy of the Test data set is  83.94518494606018
```

After increasing the dropout values, we released that the accuracy of the model was decreasing so we decreased the value of dropout and that resulted in a better accuracy.

We also found that RMSprop optimizer gives the best accuracy when the dropout value was kept as 0.04.

Finally, among the activation functions “softplus” with the combination of RELU activation functions gave the best accuracy.

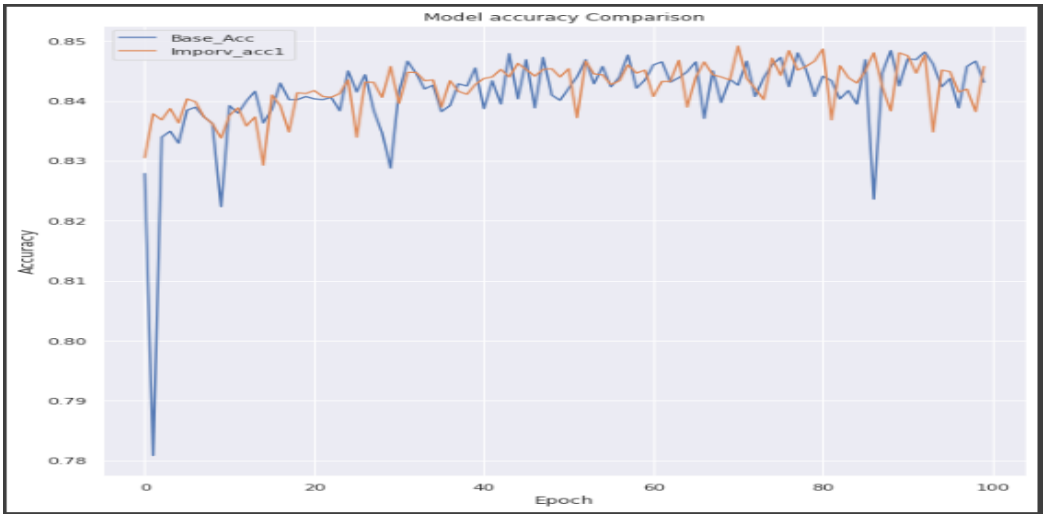
So, the final neural network model’s hyperparameters that resulted in achieving better accuracy were:

| | Final Setup |
|---|---|
| Dropout | 4% |
| Optimizer | RMSprop |
| Activation functions | Relu<br>Relu<br>Relu<br>Softplus |

We then moved on to next part which was to use different improvement methods to try and make the model more accurate.

On using the first one which is Gradient Clipping in our neural network model we found out that the accuracy of the model increased from 84.3 to 84.57% which was a good progress. The graph provided compares the accuracy of the model with improved accuracy with the base NN model.

```
505/505 [==============================] - 1s 1ms/step - loss: 0.3044 - accuracy: 0.8561
The accuracy of the Training data set is  85.60901880264282
249/249 [==============================] - 0s 1ms/step - loss: 0.3410 - accuracy: 0.8457
The accuracy of the Test data set is  84.57379937171936
```

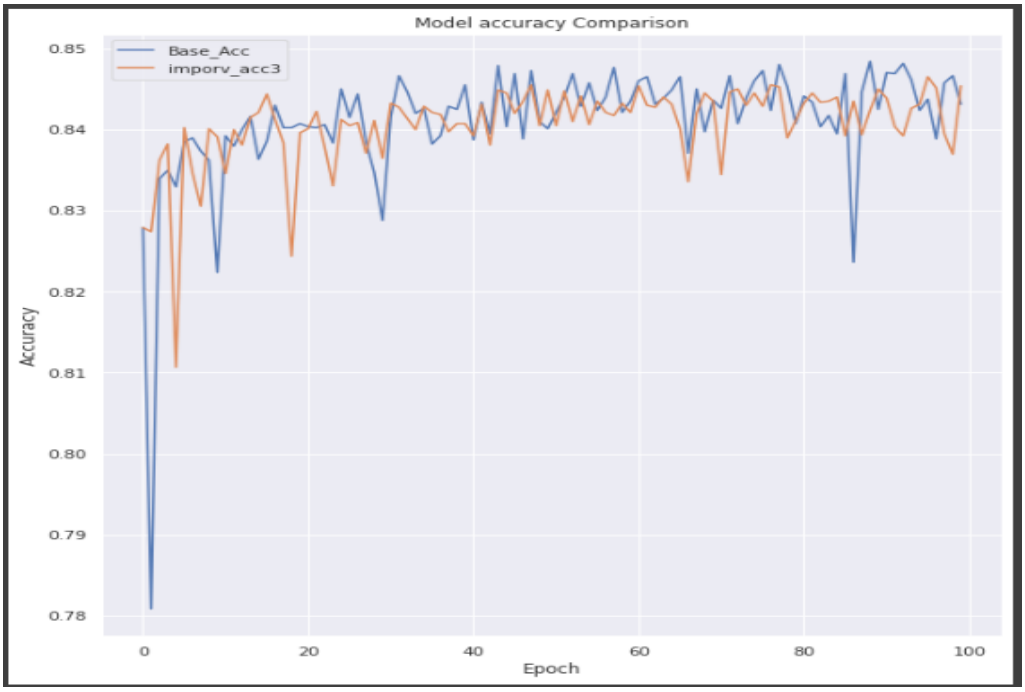



We then moved on to try the next method which is Batch Normalization.  For this method we found that the increase in accuracy of the model was not so much but still it gave an increase from 84.30 to 84.34%. The graph provided compares the accuracy of the model with improved accuracy with the base NN model.

```
505/505 [==============================] - 1s 2ms/step - loss: 0.3039 - accuracy: 0.8552
The accuracy of the Training data set is  85.51613092422485
249/249 [==============================] - 0s 2ms/step - loss: 0.3577 - accuracy: 0.8435
The accuracy of the Test data set is  84.34749841690063
```

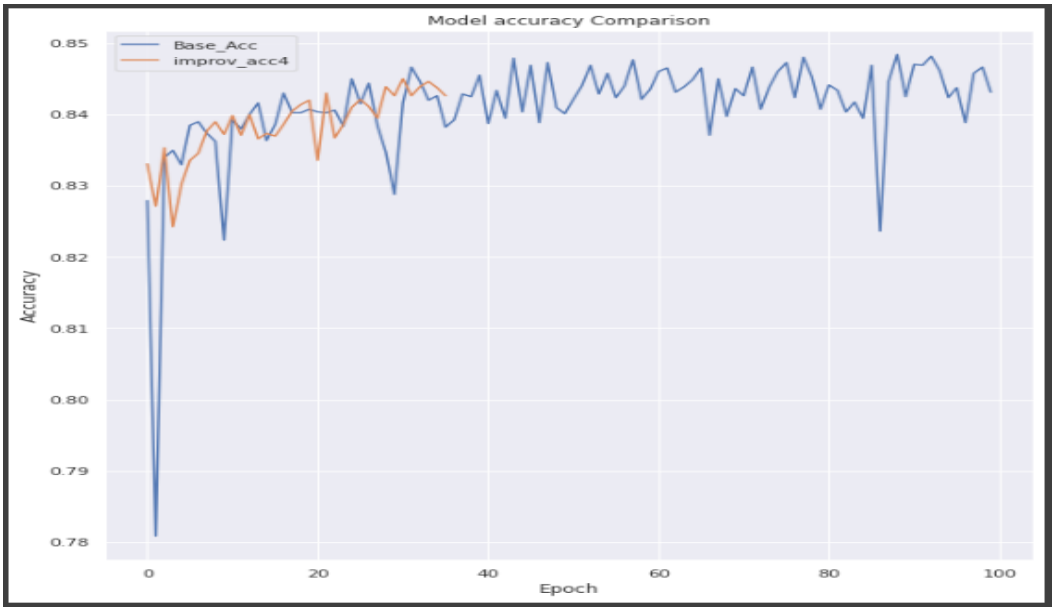

Next, we tried using Kernel Regularization which gave us an increase of accuracy from 84.30 to 84.53%. The graph provided compares the accuracy of the model with improved accuracy with the base NN model.

```
505/505 [==============================] - 1s 1ms/step - loss: 0.3255 - accuracy: 0.8547
The accuracy of the Training data set is  85.47278642654419
249/249 [==============================] - 0s 1ms/step - loss: 0.3570 - accuracy: 0.8454
The accuracy of the Test data set is  84.5360815525055
```




Finally, we tried Early Stooping on our model to see how much change in accuracy we get on using that. The model accuracy we got was 84.25% for epoch number.36 which is greater than the accuracy of the base model's accuracy at that epoch. The graph provided compares the accuracy of the model with improved accuracy with the base NN model.

```
505/505 [==============================] - 1s 1ms/step - loss: 0.3127 - accuracy: 0.8532
The accuracy of the Training data set is  85.31797528266907
249/249 [==============================] - 0s 1ms/step - loss: 0.3416 - accuracy: 0.8426
The accuracy of the Test data set is  84.25949215888977
```



From the above graph we can clearly see that the improved accuracy is giving better results as compared to our base model. If we experiment more with the hyperparameters and deep learning methods, we can increase the model accuracy even more.

In conclusion we found that our model gives the best accuracy with the inclusion of Batch Normalization in our base model and the final accuracy we get is 84.53%.

## Part-3 Building a CNN

In this part we start by taking Fashion MNIST dataset as input and start to preprocess the data like we did in the previous part to prepare the data for CNN build later. The above dataset is of Zalando's article images. It consists of 70000 samples which is divided into 60000 train and 10000 test samples. These samples consist of 28x28 greyscale images. The dataset consists of class names T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, ankle boot. Main statistics of the dataset and the visualization graph(1) are shown below.


In this visualization graph the code is taking 25 images and printing its labels with it.

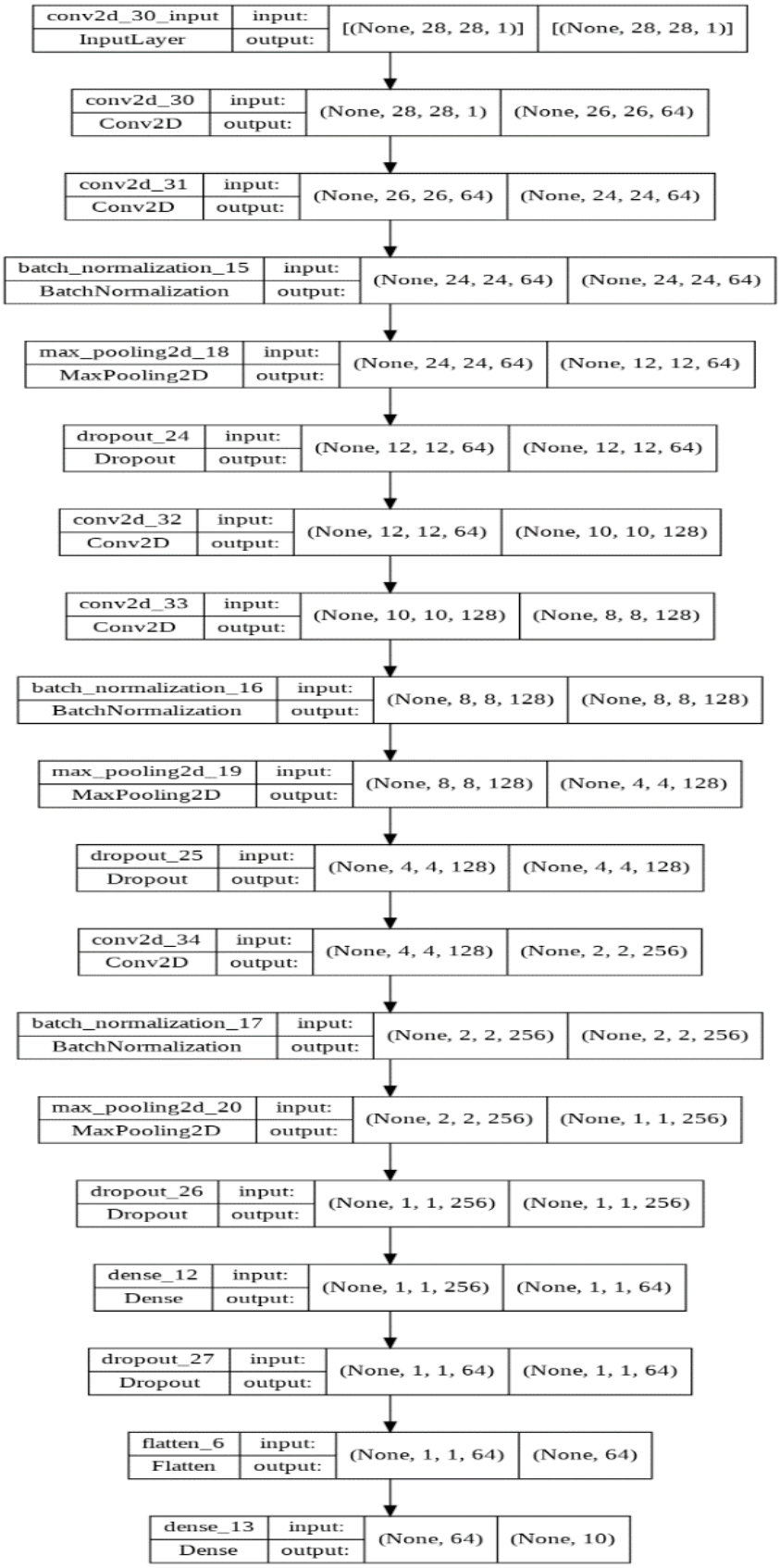
The main statistics of the data set:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 774 | 775 | 776 | 777 | 778 | 779 | 780 | 781 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 60000.000000 | 60000.000000 | 60000.000000 | 60000.000000 | 60000.000000 | 60000.000000 | 60000.000000 | 60000.000000 | 60000.000000 | 60000.000000 | ... | 60000.000000 | 60000.000000 | 60000.000000 | 60000.000000 | 60000.000000 | 60000.000000 | 60000.000000 | 60000.000000 | 60000.0 |
| mean | 0.000800 | 0.005783 | 0.030083 | 0.103800 | 0.249683 | 0.414717 | 0.821667 | 2.224733 | 5.698667 | 14.434650 | ... | 34.564367 | 23.208633 | 16.576250 | 17.831967 | 22.918850 | 17.916900 | 8.485717 | 2.706333 | 0.8 |
| std | 0.092554 | 0.249033 | 0.767868 | 2.512017 | 4.331376 | 5.827394 | 8.309935 | 14.201820 | 23.836980 | 38.204702 | ... | 57.557779 | 48.881430 | 42.044318 | 43.911297 | 51.928401 | 45.173634 | 29.448614 | 17.258682 | 9.1 |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | ... | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 |
| 25% | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | ... | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 |
| 50% | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | ... | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 |
| 75% | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | ... | 57.000000 | 8.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 |
| max | 16.000000 | 36.000000 | 119.000000 | 164.000000 | 224.000000 | 230.000000 | 221.000000 | 221.000000 | 254.000000 | 255.000000 | ... | 255.000000 | 255.000000 | 255.000000 | 255.000000 | 255.000000 | 255.000000 | 255.000000 | 255.000000 | 255.0 |

8 rows × 784 columns

We then moved on to preprocess the dataset by just reshaping the images. Then divide it into training and testing datasets using various libraries used in Part-1.

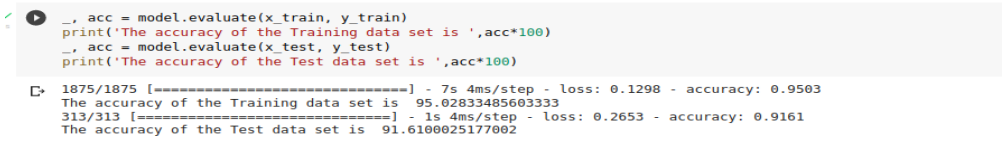
We then developed a CNN using the preprocessed data we got to make a deep learning model with 5 hidden layers, with 64 input neurons and 10 output neurons. In the model we have used RELU and softmax activation functions initially which are updated in later parts for better accuracy. We have used 3 as the kernel size for our model in all layers. We do not have any padding in our model. RELU activation function is used in the hidden layer and softmax is used in the output layer. Below is the architecture of the CNN model.(After getting accuracy greater than 92% when we used Batch Normalization)


conv2d_30_input | InputLayer | input: | output: | [(None, 28, 28, 1)] | [(None, 28, 28, 1)]
conv2d_30 | Conv2D | input: | output: | (None, 28, 28, 1) | (None, 26, 26, 64)
conv2d_31 | Conv2D | input: | output: | (None, 26, 26, 64) | (None, 24, 24, 64)
batch_normalization_15 | BatchNormalization | input: | output: | (None, 24, 24, 64) | (None, 24, 24, 64)
max_pooling2d_18 | MaxPooling2D | input: | output: | (None, 24, 24, 64) | (None, 12, 12, 64)
dropout_24 | Dropout | input: | output: | (None, 12, 12, 64) | (None, 12, 12, 64)
conv2d_32 | Conv2D | input: | output: | (None, 12, 12, 64) | (None, 10, 10, 128)
conv2d_33 | Conv2D | input: | output: | (None, 10, 10, 128) | (None, 8, 8, 128)
batch_normalization_16 | BatchNormalization | input: | output: | (None, 8, 8, 128) | (None, 8, 8, 128)
max_pooling2d_19 | MaxPooling2D | input: | output: | (None, 8, 8, 128) | (None, 4, 4, 128)
dropout_25 | Dropout | input: | output: | (None, 4, 4, 128) | (None, 4, 4, 128)
conv2d_34 | Conv2D | input: | output: | (None, 4, 4, 128) | (None, 2, 2, 256)
batch_normalization_17 | BatchNormalization | input: | output: | (None, 2, 2, 256) | (None, 2, 2, 256)
max_pooling2d_20 | MaxPooling2D | input: | output: | (None, 2, 2, 256) | (None, 1, 1, 256)
dropout_26 | Dropout | input: | output: | (None, 1, 1, 256) | (None, 1, 1, 256)
dense_12 | Dense | input: | output: | (None, 1, 1, 256) | (None, 1, 1, 64)
dropout_27 | Dropout | input: | output: | (None, 1, 1, 64) | (None, 1, 1, 64)
flatten_6 | Flatten | input: | output: | (None, 1, 1, 64) | (None, 64)
dense_13 | Dense | input: | output: | (None, 64) | (None, 10)

And the model summery is –

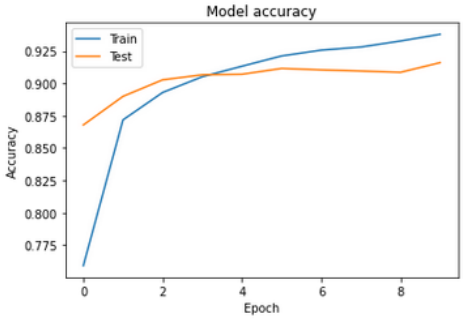
```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_30 (Conv2D)          (None, 26, 26, 64)        640

 conv2d_31 (Conv2D)          (None, 24, 24, 64)        36928

 batch_normalization_15 (Bat  (None, 24, 24, 64)       256
 chNormalization)

 max_pooling2d_18 (MaxPoolin  (None, 12, 12, 64)       0
 g2D)

 dropout_24 (Dropout)        (None, 12, 12, 64)        0

 conv2d_32 (Conv2D)          (None, 10, 10, 128)       73856

 conv2d_33 (Conv2D)          (None, 8, 8, 128)         147584

 batch_normalization_16 (Bat  (None, 8, 8, 128)        512
 chNormalization)

 max_pooling2d_19 (MaxPoolin  (None, 4, 4, 128)        0
 g2D)

 dropout_25 (Dropout)        (None, 4, 4, 128)         0

 conv2d_34 (Conv2D)          (None, 2, 2, 256)         295168

 batch_normalization_17 (Bat  (None, 2, 2, 256)        1024
 chNormalization)

 max_pooling2d_20 (MaxPoolin  (None, 1, 1, 256)        0
 g2D)

 dropout_26 (Dropout)        (None, 1, 1, 256)         0

 dense_12 (Dense)            (None, 1, 1, 64)          16448

 dropout_27 (Dropout)        (None, 1, 1, 64)          0

 flatten_6 (Flatten)         (None, 64)                0

 dense_13 (Dense)            (None, 10)                650

=================================================================
Total params: 573,066
Trainable params: 572,170
Non-trainable params: 896
_________________________________________________________________
```

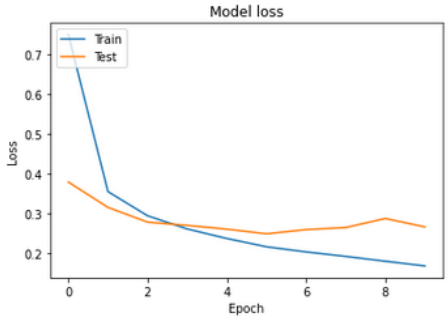
The output accuracy graph and accuracy of our basic CNN model is shown below-



```
_, acc = model.evaluate(x_train, y_train)
print('The accuracy of the Training data set is ',acc*100)
_, acc = model.evaluate(x_test, y_test)
print('The accuracy of the Test data set is ',acc*100)
```

```
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1298 - accuracy: 0.9503
The accuracy of the Training data set is  95.02833485603333
313/313 [==============================] - 1s 4ms/step - loss: 0.2653 - accuracy: 0.9161
The accuracy of the Test data set is  91.6100025177002
```



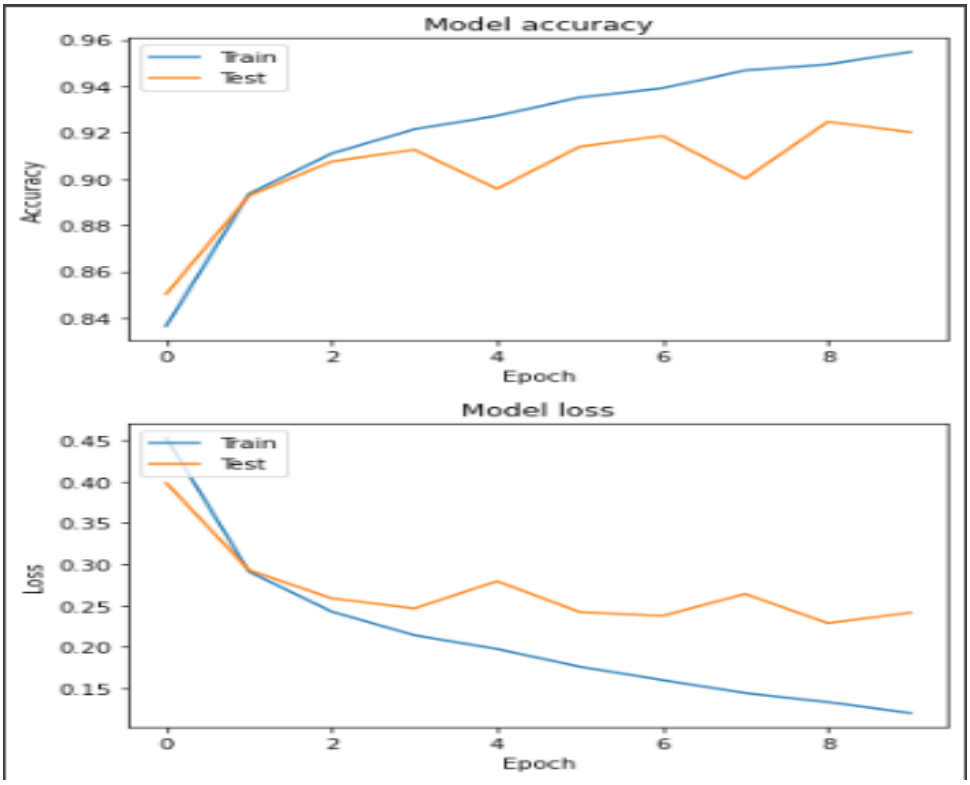

Visualization graph(2)


This visualization graph is giving us 25 correctly predicted images with its labels.

We can see that the final accuracy of our model is coming out to be 91.61% which is a relatively good start position to build up.

We then applied improvement methods on our existing model. By doing Batch Normalization helped us in getting a more accurate model. Below is the final accuracy and graphs for our base CNN model which will be used in the following parts.

```
1875/1875 [==============================] - 7s 3ms/step - loss: 0.1036 - accuracy: 0.9608
The accuracy of the Training data set is  96.07666730880737
313/313 [==============================] - 1s 4ms/step - loss: 0.2413 - accuracy: 0.9201
The accuracy of the Test data set is  92.00999736785889
```




We can clearly see that now the accuracy is increased to **92. 01%**.

## Part – 4 Optimizing CNN and Data Augmentation –

We now move on to increase the accuracy of our model by experimenting with various hyperparameters in our model. For this we used different dropout values, activation functions and optimizers to get the best possible base model to further work on.

Below is the table with different Optimizer values:

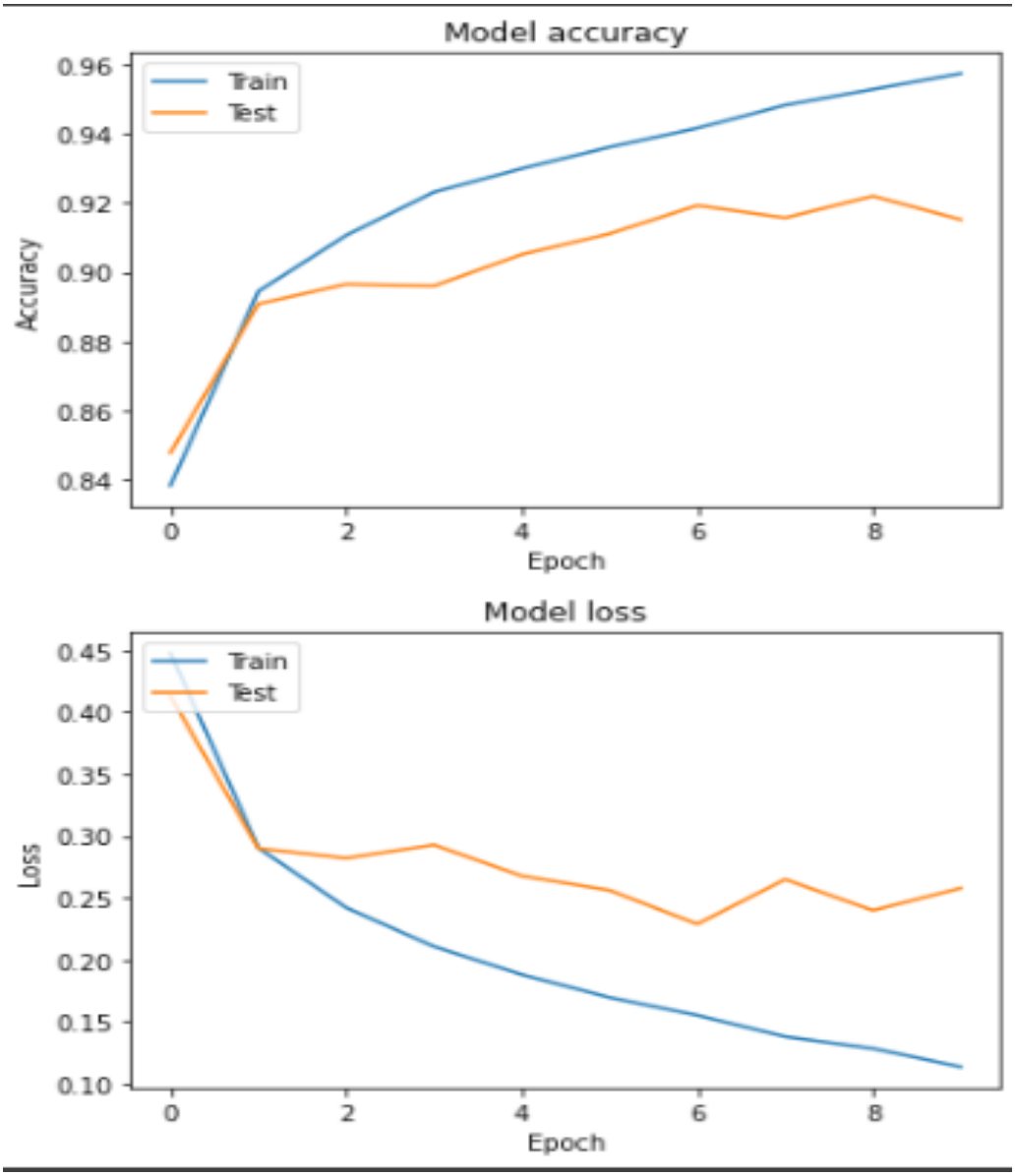
| | Setup-1 | Accuracy | Setup-2 | Accuracy | Setup-3 | Accuracy |
|---|---|---|---|---|---|---|
| Dropout | 7% | 91.51% | 7% | 92.11% | 7% | 88.52% |
| Optimizer | NADAM | | RMSprop | | Adagrad | |
| Activatio | RELU | | RELU | | RELU | |

| | | | | | | |
|---|---|---|---|---|---|---|
| n functions | And Softmax | | And Softmax | | And Softmax | |

Below are the graph and model accuracy for all setups:

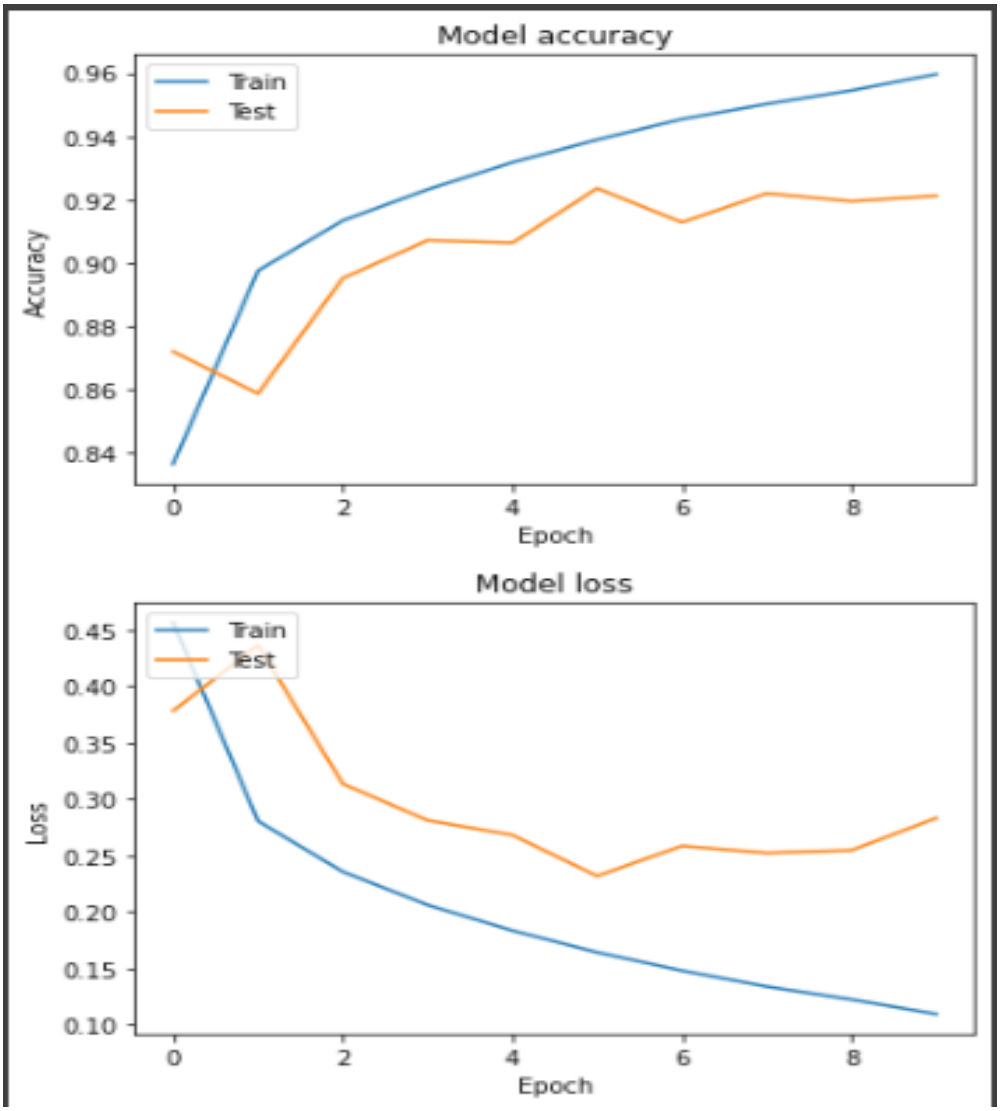
Setup – 1

```
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1008 - accuracy: 0.9619
The accuracy of the Training data set is  96.1899995803833
313/313 [==============================] - 1s 4ms/step - loss: 0.2578 - accuracy: 0.9152
The accuracy of the Test data set is  91.51999950408936
```



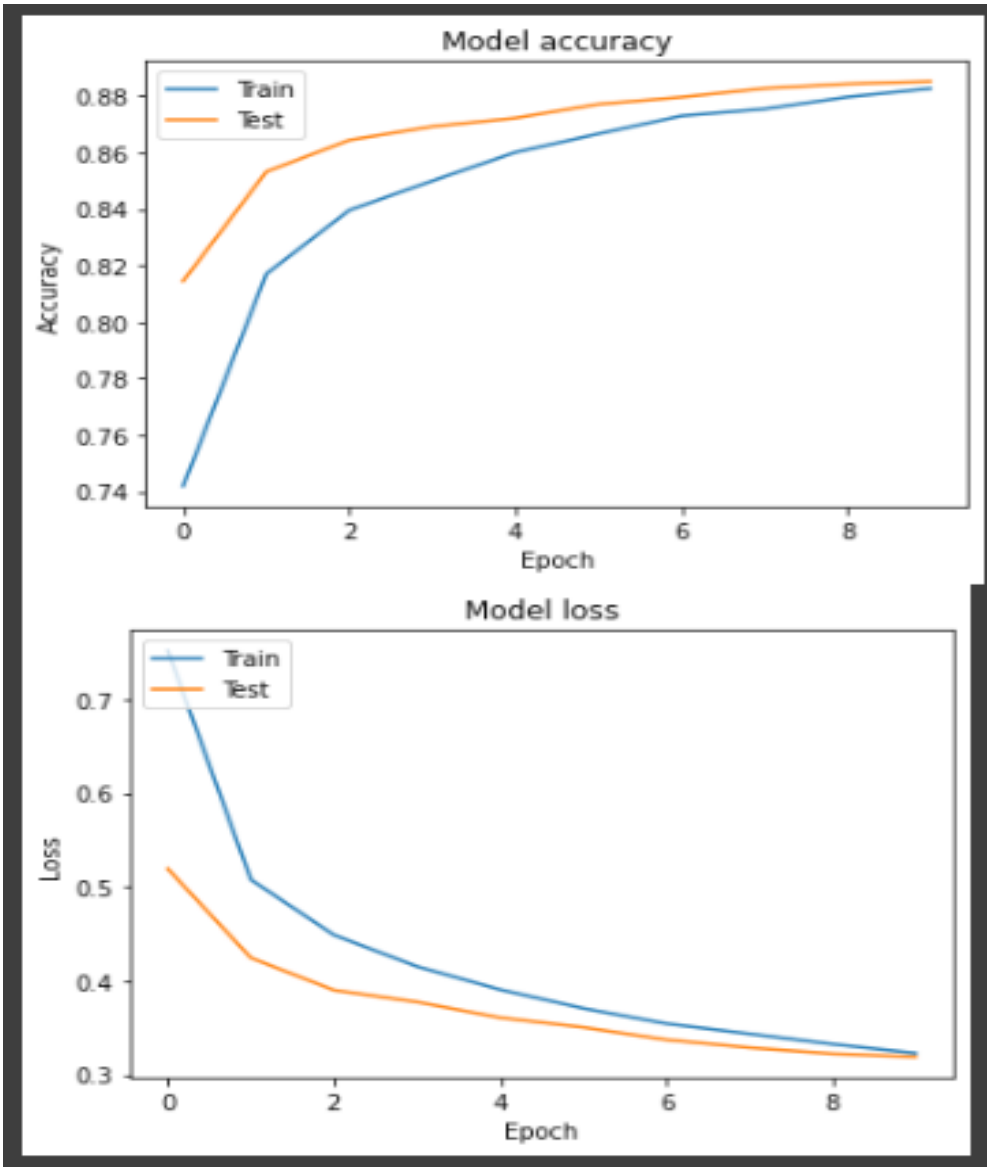

Setup – 2

```
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0786 - accuracy: 0.9708
The accuracy of the Training data set is  97.079998254776
313/313 [==============================] - 1s 4ms/step - loss: 0.2833 - accuracy: 0.9211
The accuracy of the Test data set is  92.11000204086304
```




Setup – 3

```
1875/1875 [==============================] - 7s 3ms/step - loss: 0.2554 - accuracy: 0.9077
The accuracy of the Training data set is  90.77333211898804
313/313 [==============================] - 1s 4ms/step - loss: 0.3191 - accuracy: 0.8852
The accuracy of the Test data set is  88.5200023651123
```



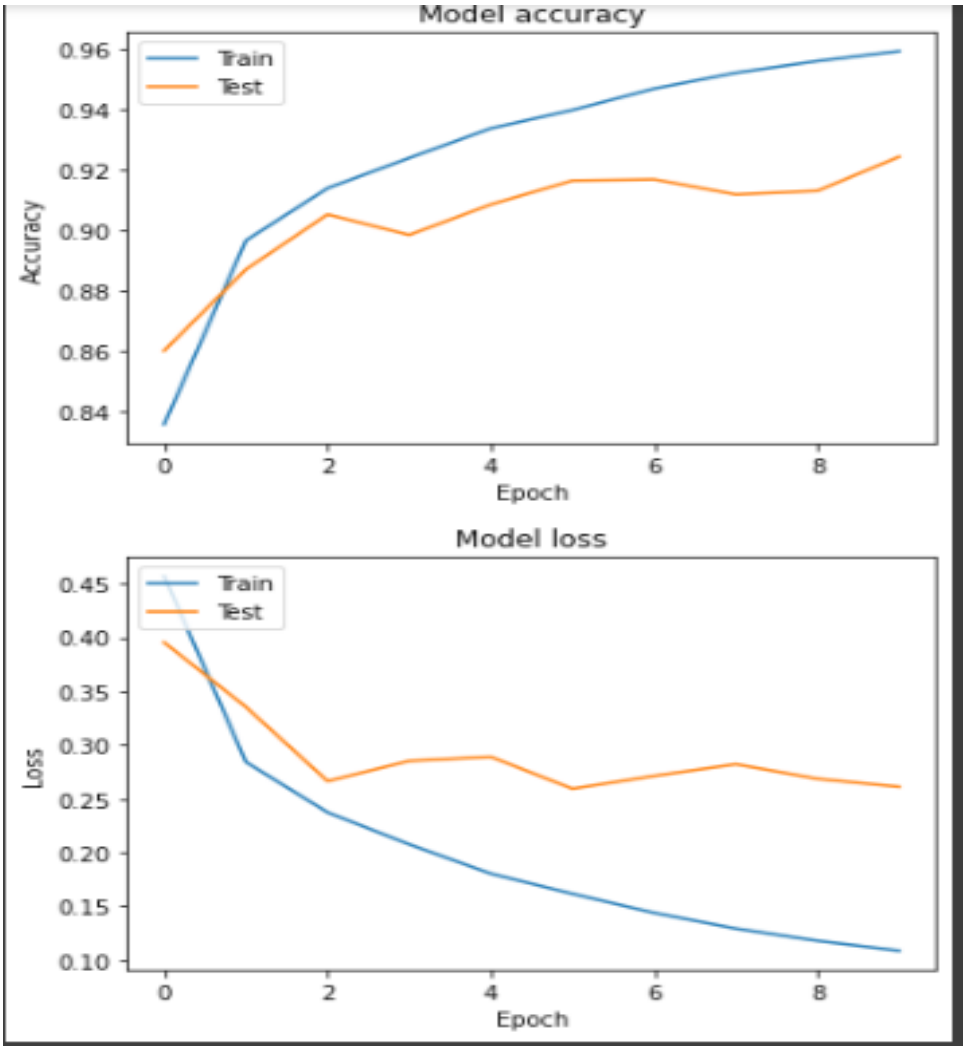

From the above results we can clearly see that RMSprop is the best optimizer to use for our model. It gives the accuracy of 92.11%.

Next, we move on to different activation functions. For this we tried to use 3 different activation functions, softmax, softplus and sigmoid with the combination of RELU in all the setups. Below is the table of the setups followed by the model accuracy and graphs.

| | Setup-1 | Accuracy | Setup-2 | Accuracy | Setup-3 | Accuracy |
|---|---|---|---|---|---|---|
| Dropout | 7% | 92.44% | 7% | 92.12% | 7% | 91.87% |
| Optimizer | RMSprop | | RMSprop | | RMSprop | |
| Activation functions | RELU And Softmax | | RELU And Softplus | | RELU And Sigmoid | |

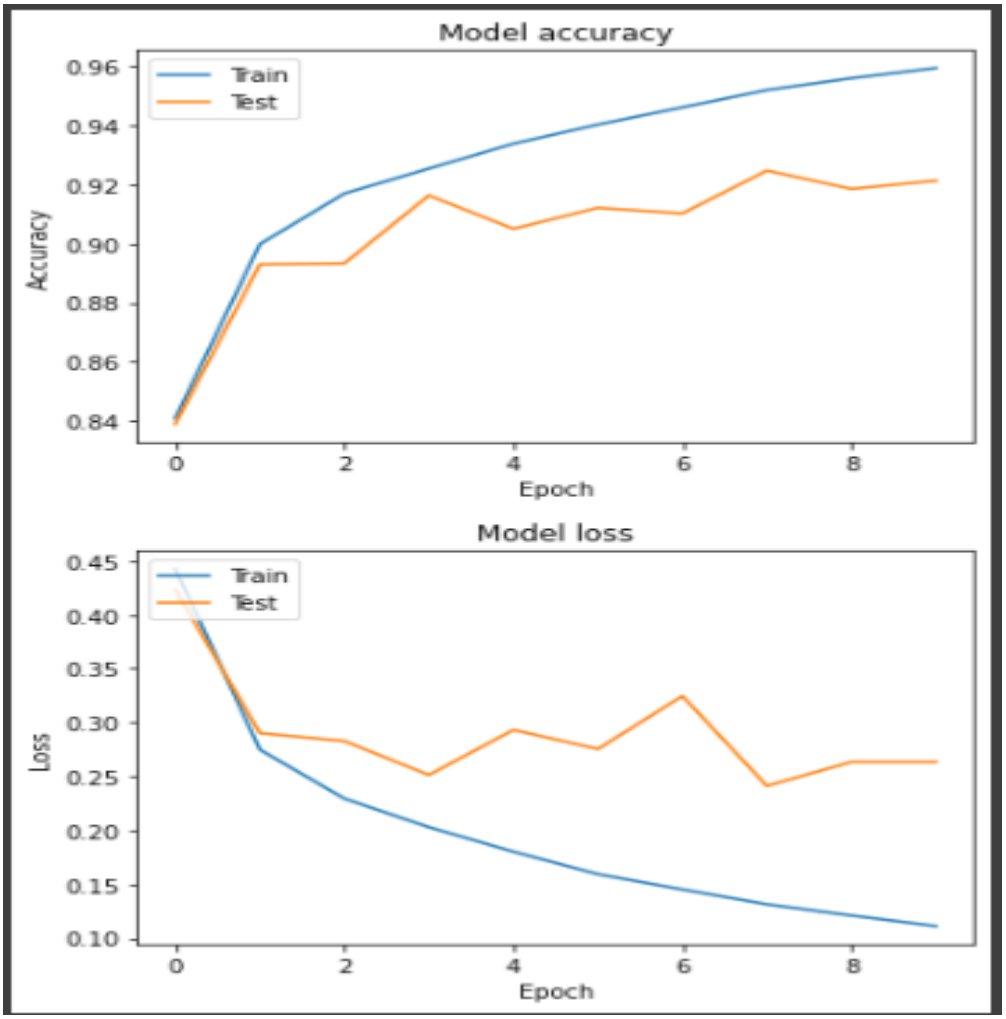
Setup – 1

```
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0793 - accuracy: 0.9701
The accuracy of the Training data set is  97.00999855995178
313/313 [==============================] - 1s 4ms/step - loss: 0.2610 - accuracy: 0.9245
The accuracy of the Test data set is  92.44999885559082
```



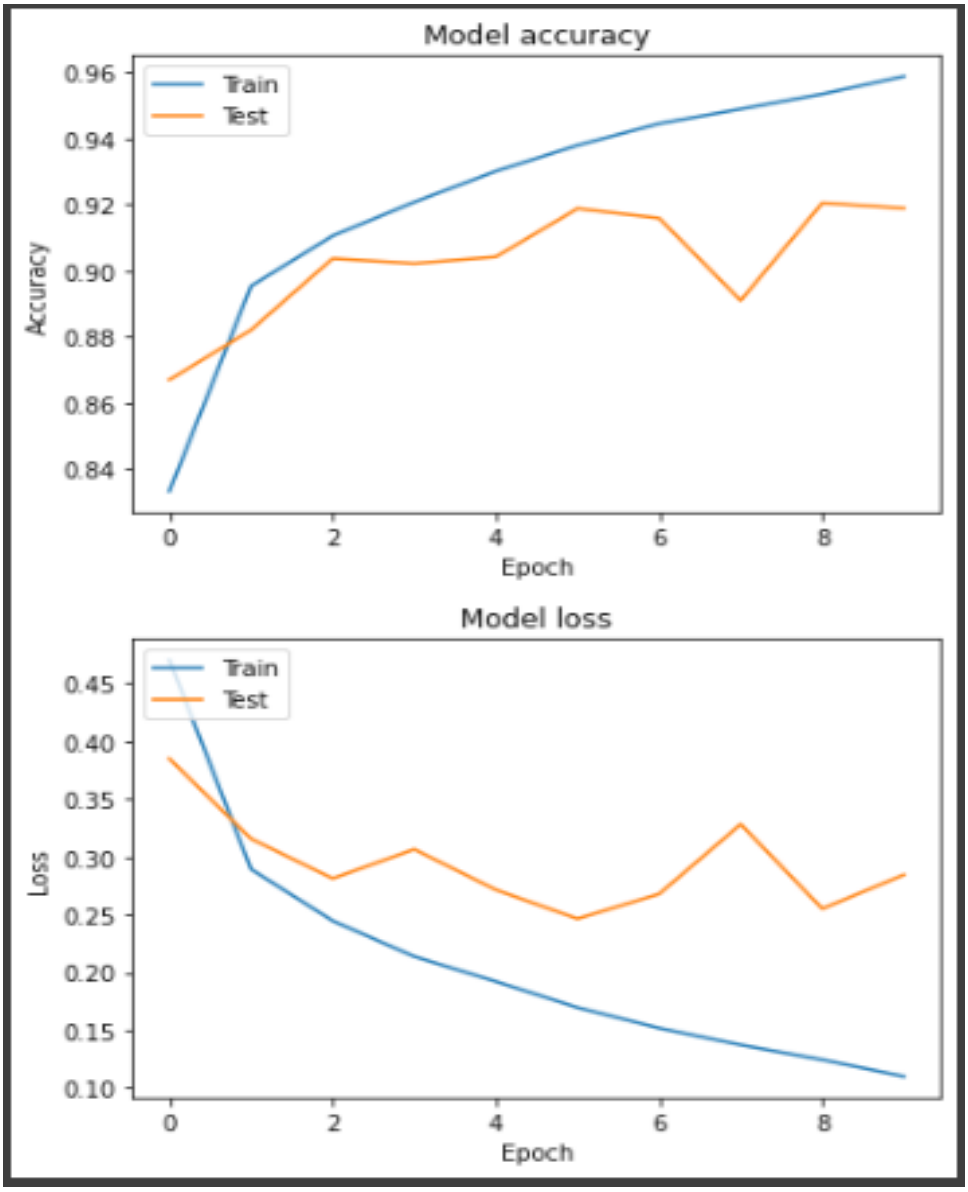

Setup – 2

```
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0834 - accuracy: 0.9693
The accuracy of the Training data set is  96.92833423614502
313/313 [==============================] - 1s 4ms/step - loss: 0.2634 - accuracy: 0.9213
The accuracy of the Test data set is  92.12999939918518
```




Setup – 3

```
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0945 - accuracy: 0.9650
The accuracy of the Training data set is  96.4983344078064
313/313 [==============================] - 1s 4ms/step - loss: 0.2844 - accuracy: 0.9188
The accuracy of the Test data set is  91.87999963760376
```



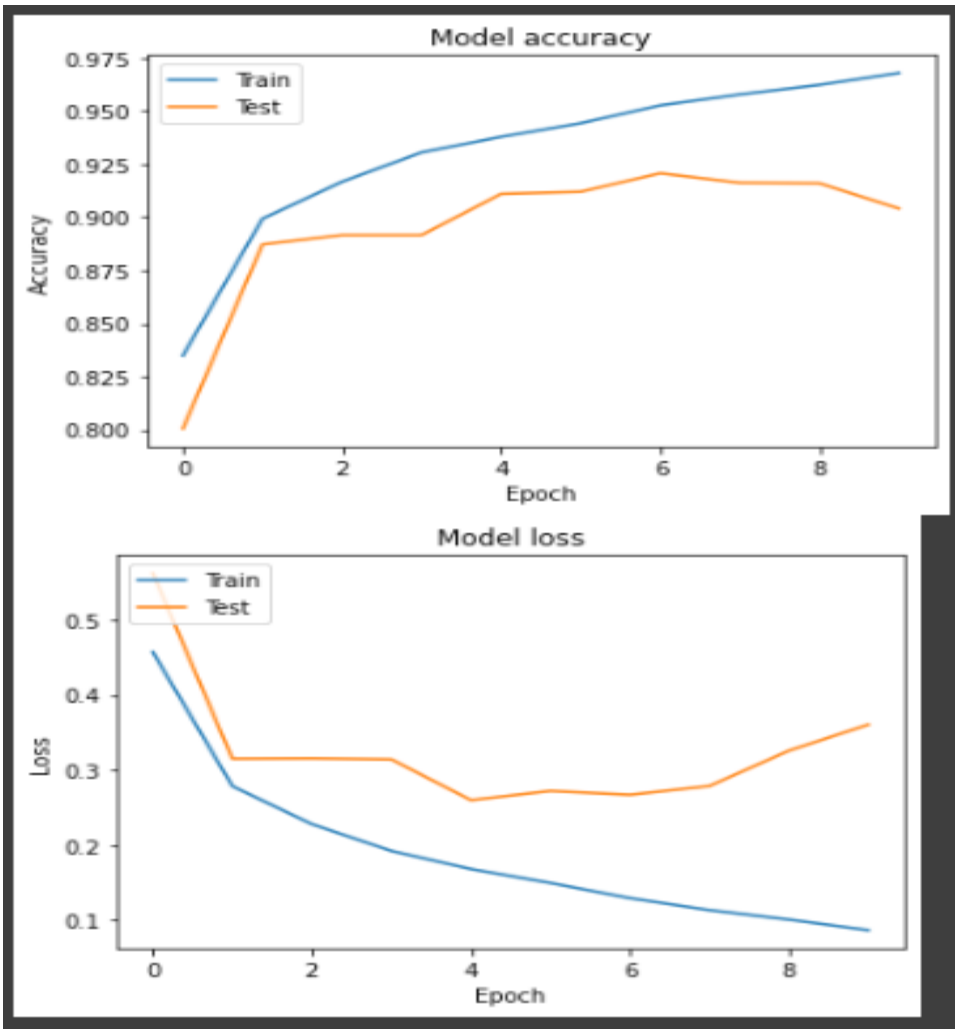

From the above observations we can conclude that softmax activation function gives the best possible accuracy for our model.

We then move on to the next part which is using different dropout values to find the best possible accuracy of our model. We did this by using 4, 10 and 7 % dropout values. The following table gives the details for the 3 setups, and it is then followed by the model accuracy and graphs of the setups.

| | Setup-1 | Accuracy | Setup-2 | Accuracy | Setup-3 | Accuracy |
|---|---|---|---|---|---|---|
| Dropout | 4% | 92.44% | 10% | 92.12% | 7% | 91.87% |
| Optimizer | RMSprop | | RMSprop | | RMSprop | |
| Activation functions | RELU And Softmax | | RELU And Softmax | | RELU And Softmax | |

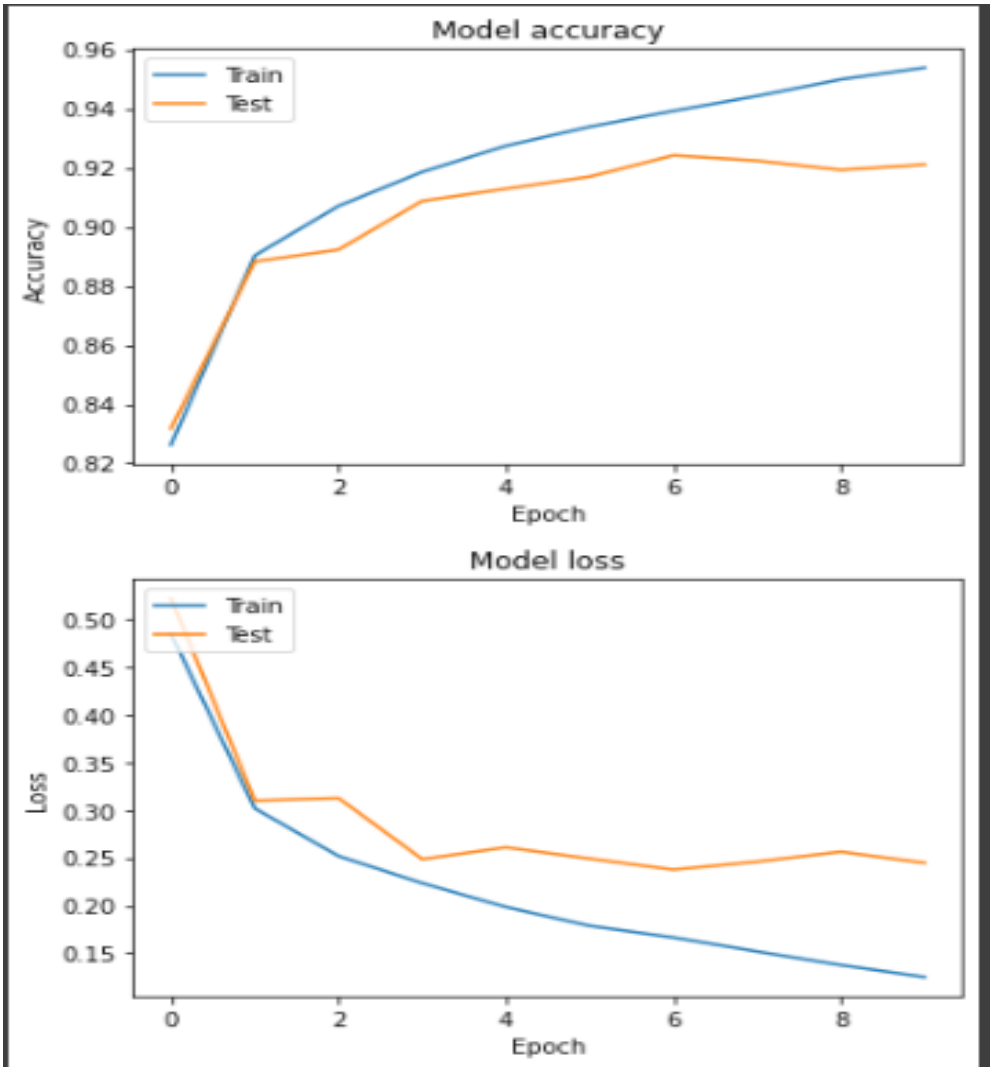
Setup – 1

```
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1143 - accuracy: 0.9570
The accuracy of the Training data set is  95.701664686203
313/313 [==============================] - 1s 4ms/step - loss: 0.3604 - accuracy: 0.9041
The accuracy of the Test data set is  90.41000008583069
```



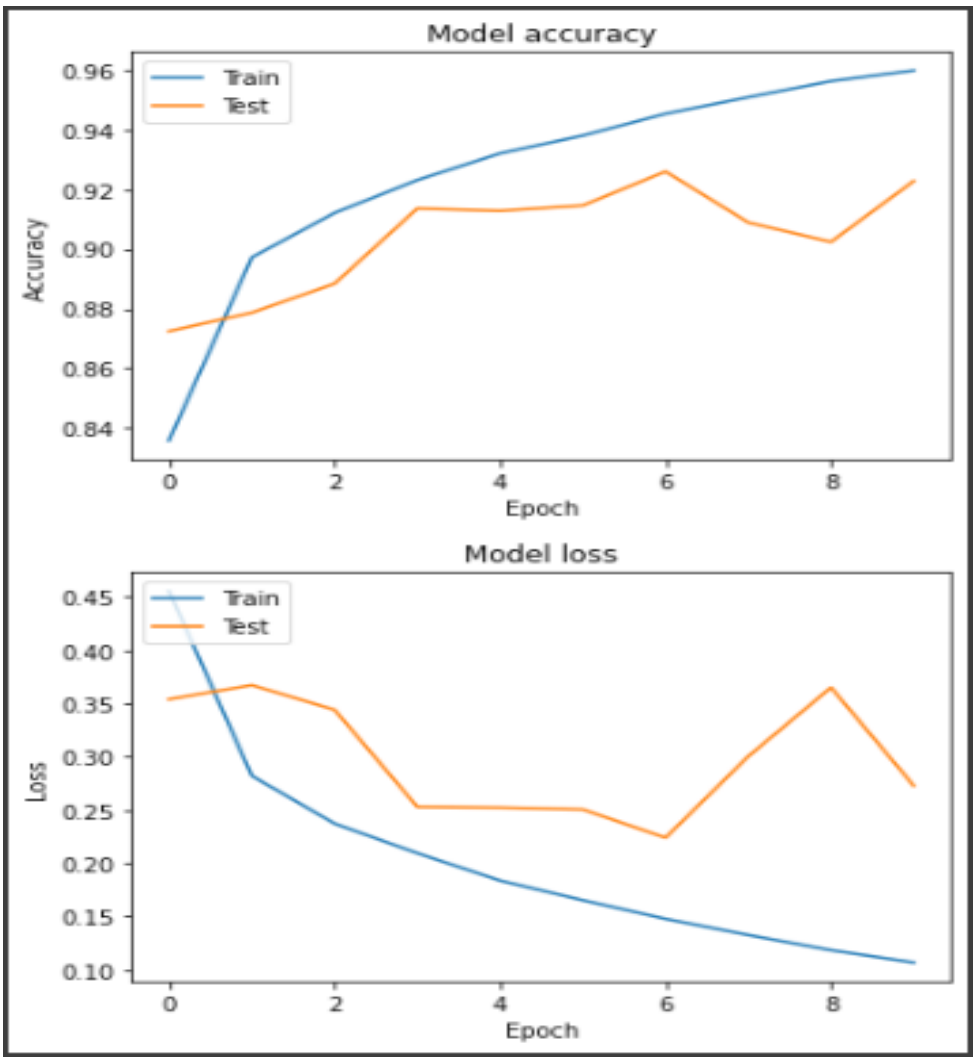

Setup – 2

```
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0831 - accuracy: 0.9699
The accuracy of the Training data set is  96.99333310127258
313/313 [==============================] - 1s 4ms/step - loss: 0.2446 - accuracy: 0.9210
The accuracy of the Test data set is  92.10000038146973
```




Setup – 3

```
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0703 - accuracy: 0.9746
The accuracy of the Training data set is  97.45833277702332
313/313 [==============================] - 1s 4ms/step - loss: 0.2728 - accuracy: 0.9228
The accuracy of the Test data set is  92.28000044822693
```




From the above results we can clearly see that 7% is the best dropout value for our CNN model which makes the final CNN model hyperparameters as follows

| | Final Setup |
|---|---|
| Dropout | 7% |
| Optimizer | RMSprop |
| Activation functions | RELU and Softmax |

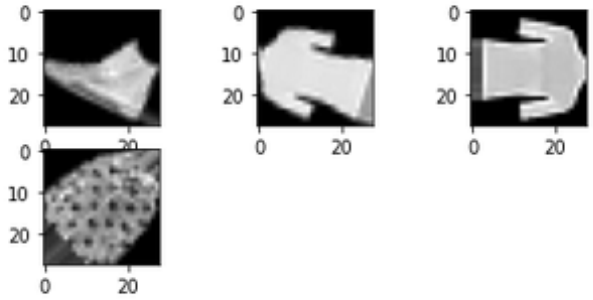
With the above hyperparameters we get the final accuracy as **92.61%** which has improved our accuracy from the base model which was 92.01%

```
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0745 - accuracy: 0.9721
The accuracy of the Training data set is  97.20666408538818
313/313 [==============================] - 1s 4ms/step - loss: 0.2573 - accuracy: 0.9262
The accuracy of the Test data set is  92.61999726295471
```

Data Augmentation:

Now we move on to data augmentation step which is used to increase the dataset by x4 to help us improve the accuracy of our model. There are a lot of methods for data augmentation, we in this case have used rotation augmentation. Rotation augmentation is generally useful because it augments the dataset by changing the angles in the objects in the dataset. This means random images are randomly rotated either clockwise or anticlockwise by 90 degree to augment the dataset.

Visualization Graph(3):

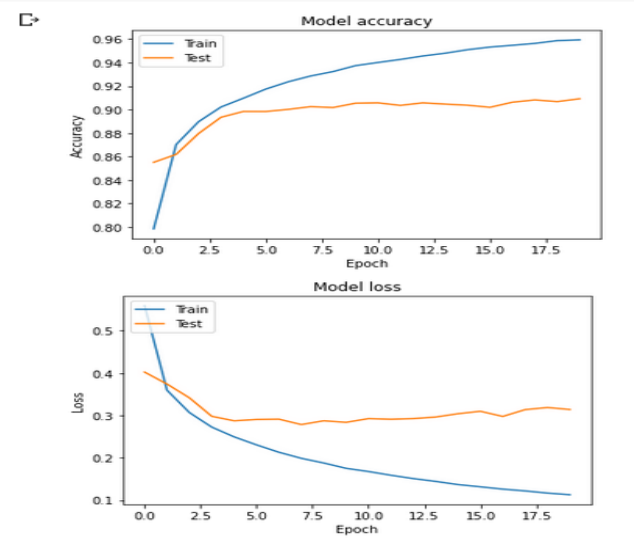


In this visualization graph it will show 4 images rotated by 90 degrees.

The output accuracy,loss graphs and accuracy of our augmented CNN model is shown below:

```
] _, acc = model.evaluate(x_train, y_train)
  print('The accuracy of the Training data set is ',acc*100)
  _, acc = model.evaluate(x_test, y_test)
  print('The accuracy of the Test data set is ',acc*100)
```

```
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0427 - accuracy: 0.9851
The accuracy of the Training data set is  98.51499795913696
313/313 [==============================] - 2s 5ms/step - loss: 0.2619 - accuracy: 0.9284
The accuracy of the Test data set is  92.83999800682068
```



We can see the accuracy after running the model after data augmentation by x4 is **92.83%**. The  accuracy of the Final CNN model before augmentation was 92.61. The accuracy after data augmentation increases because we expand our data set and provide enough training to develop a reliable model. Also, it reduces over-fitting hence giving an improved accuracy.

| Team Member | Assignment Part | Contribution(%) |
|---|---|---|
| Pratik Malani | Parts 1-4 and Bonus | 50% |
| Dhruv Patel | Parts 1-4 and Bonus | 50% |

**References:**


Stackoverflow.com
Geeksforgeeks.org
Tutorialgateway.org
towardsdatascience.com
https://pandas.pydata.org/docs/reference/frame.html
https://matplotlib.org/stable/tutorials/introductory/usage.html#sphx-glr-tutorials-introductory-usagepy
https://scikit-learn.org/stable/
https://keras.io/
https://www.tensorflow.org/
https://www.analyticsvidhya.com/blog/2020/08/image-augmentation-on-the-fly-using-keras-imagedatagenerator/
https://machinelearningmastery.com/implement-perceptron-algorithm-scratch-python/
https://valohai.com/blog/data-augmentation/