UNIVERSITY OF TEHRAN

COLLEGE OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NEURAL NETWORK

ASSIGNMENT#2

MOHAMMAD HEYDARI

810197494

UNDER SUPERVISION OF:

DR. AHMAD KALHOR

ASSISTANT PROFESSOR

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

UNIVERSITY OF TEHRAN

Mar. 2022

1 CONTENTS

2 QUESTION #1

2.1 MULTI-LAYER PERCEPTRON (CLASSIFICATION)

In this part we intend to implement a classifier based on multi-layer perceptron concept which is used to classify Cifar10 dataset.

As we know the dataset consists 60000 rows which is prepared in ten category.

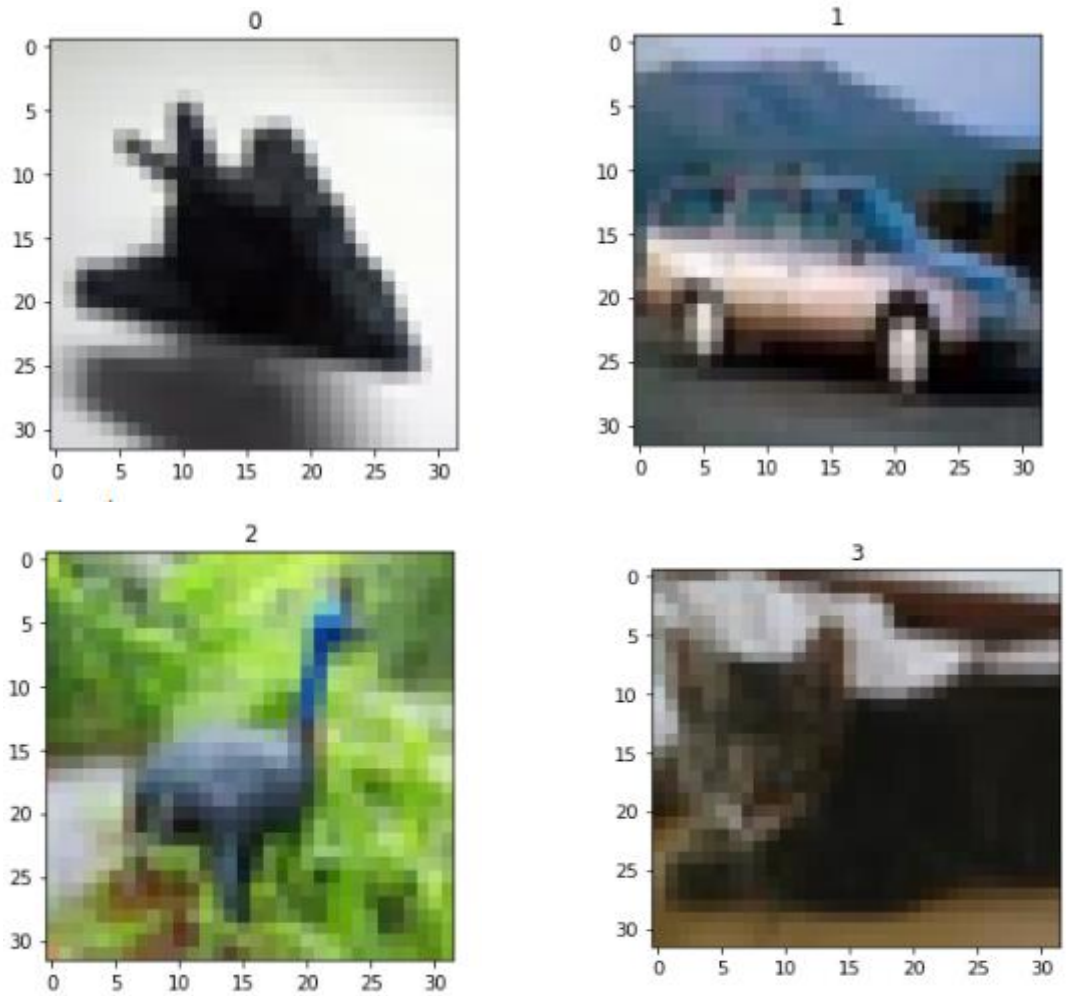


Figure 1: Representation of Dataset

Figure 2: Representation of Dataset

2.1.1 WHAT'S THE OPTIMAL MACHINE LEARNING DATA SPLIT RATIO AND HOW TO ACHIEVE IT?

The creation of different samples and splits in the dataset helps us judge the true model performance.

The dataset split ratio depends on the number of samples present in the dataset and the model.

Some common inferences that can be derived on dataset split include:

1) If there are several hyperparameters to tune, the machine learning model requires a larger validation set to optimize the model performance. Similarly, if the model has fewer or no hyperparameters, it would be easy to validate the model using a small set of data.

2) If a model use case is such that a false prediction can drastically hamper the model performance—like falsely predicting cancer—it's better to validate the model after each epoch to make the model learn varied scenarios.

3) With the increase in the dimension/features of the data, the hyperparameters of the neural network also increase making the model more complex. In these scenarios, a large split of data should be kept in training set with a validation set.

However, there are two major concerns while deciding on the optimum split:

If there is less training data, the machine learning model will show high variance in training.

- With less testing data/validation data, your model evaluation/model performance statistic will have greater variance.

- With less testing data/validation data, your model evaluation/model performance statistic will have greater variance.

Essentially, we need to come up with an optimum split that suits the need of the dataset/model.

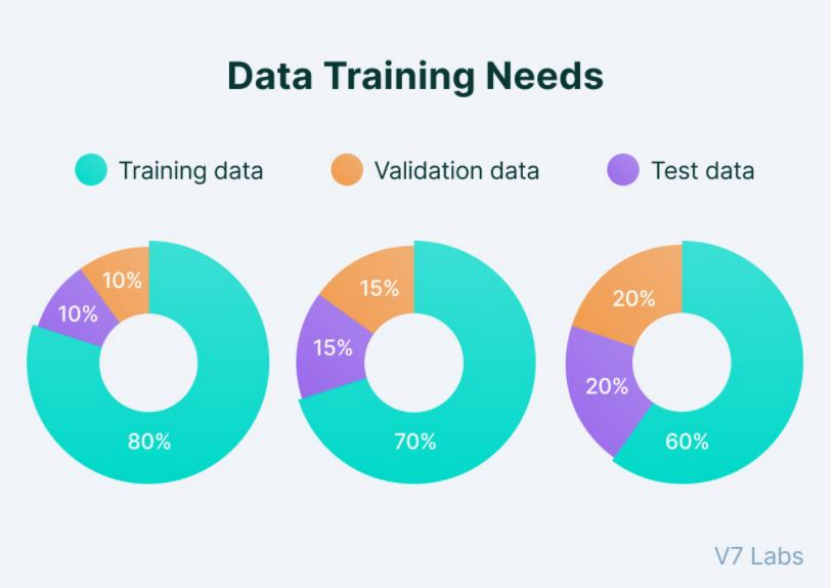


Figure 1: Data-Split standard percentage

Finally, here's a recap of everything we've learned:

Training data is the set of the data on which the actual training takes place. Validation split helps to improve the model performance by fine-tuning the model after each epoch. The test set informs us about the final accuracy of the model after completing the training phase.

The training set should not be too small; else, the model will not have enough data to learn. On the other hand, if the validation set is too small, then the evaluation metrics like accuracy, precision, recall, and F1 score will have large variance and will not lead to the proper tuning of the model.

In general, putting 80% of the data in the training set, 10% in the validation set, and 10% in the test set is a good split to start with.

The optimum split of the test, validation, and train set depends upon factors such as the use case, the structure of the model, dimension of the data, etc.

2.1.2 THEORETICAL NOTES ABOUT CONFUSION MATRIX, F1-SCORE, RECALL, PRECISION:

Confusion-matrix:

As we know A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class.

This is the key to the confusion matrix. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the

errors being made by our classifier but more importantly the types of errors that are being made. It is this breakdown that overcomes the limitation of using classification accuracy alone.

"True positive" for correctly predicted event values, and in our case it's **2**

"False positive" for incorrectly predicted event values, and in our case it's **1**

"True negative" for correctly predicted no-event values, and in our case it's **1**

"False negative" for incorrectly predicted no-event values, and in our case it's **1**

Confusion matrix abbreviated below:

$$\begin{bmatrix} TP = 2 & TN = 1 \\ FP = 1 & FN = 1 \end{bmatrix}$$

And at the end, Accuracy is one metric for evaluating classification models. Informally, accuracy is the fraction of predictions our model got right. Formally, accuracy has the following definition:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

For binary classification, accuracy can also be calculated in terms of positives and negatives as follows:

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

Where TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives.

Recall:

Intuitively, we know that proclaiming all data points as negative (not a terrorist) in the terrorist detection problem isn't helpful, and, instead, we should focus on identifying the positive cases. The metric our intuition tells us we should maximize is known in statistics as recall, or the ability of a model to find all the relevant cases within a data set. The technical definition of recall is the number of true positives divided by the number of true positives plus the number of false negatives.

Recall: The ability of a model to find all the relevant cases within a data set. Mathematically, we define recall as the number of true positives divided by the number of true positives plus the number of false negatives.

$$recall = \frac{true\ positives}{true\ positives\ + false\ negatives} = \frac{terrorists\ correctly\ identified}{terrorists\ correctly\ identified + terrorists\ incorrectly\ labeled\ as\ not\ terrorists}$$

Precision:

Again, we know intuitively that a model labelling 100 percent of passengers as terrorists is probably not useful because we would have to ban every single person from flying. Statistics provides us with the vocabulary to express our intuition: this new model would suffer from low precision or the ability of a classification model to identify only the relevant data points.

As we increase precision, we decrease recall and vice-versa.

Precision: The ability of a classification model to identify only the relevant data points. Mathematically, precision the number of true positives divided by the number of true positives plus the number of false positives.

$$precision = \frac{true\ positives}{true\ positives + false\ positives} = \frac{terrorists\ correctly\ identified}{terrorists\ correctly\ identified + individuals\ incorrectly\ labeled\ as\ terroists}$$

F1-Score:

The F1-score combines the precision and recall of a classifier into a single metric by taking their harmonic mean. It is primarily used to compare the performance of two classifiers. Suppose that classifier A has a higher recall, and classifier B has higher precision. In this case, the F1-scores for both the classifiers can be used to determine which one produces better results.

The F1-score of a classification model is calculated as follows:

$$\frac{2(P * R)}{P + R}$$

P = the precision

R = the recall of the classification model

$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{tp}}{\mathrm{tp} + \frac{1}{2}(\mathrm{fp} + \mathrm{fn})}.$$

In the second part we are going to do some calculations in case of pre-processing consisting normalizing, grey-scaling and reshape the size of images.

Further you can find the whole process which is done in these cases.

2.1.3 NUMBER OF LAYERS ANALYSE:

In this part we have chosen the batch-size equal to 64 as of a general standard.

Number of hidden layer neurons : 128

Architecture of neural net :

In this part I have used **four layers** neural net which consist **two hidden layers**. At the final layer I have used **softmax** as of our activation-function and **ReLu** at the hidden layers.

I have used **512 neurons** in first layer , **128 neurons** in second layer and **128 neurons** in third layer and **10 neurons** at the final layer.

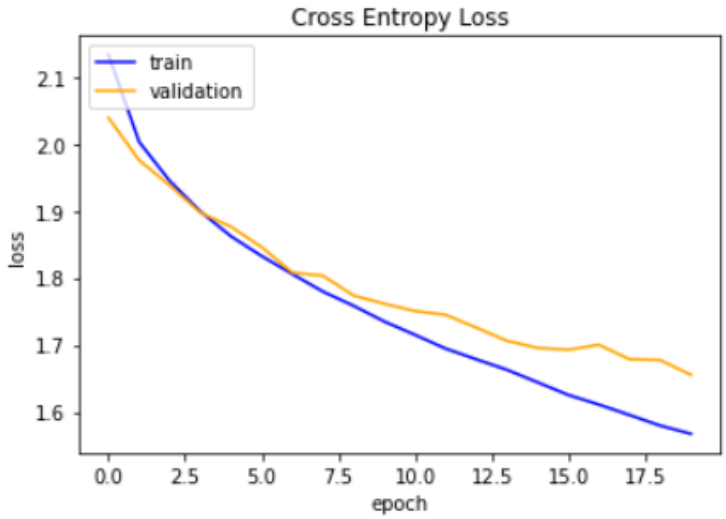
Results:

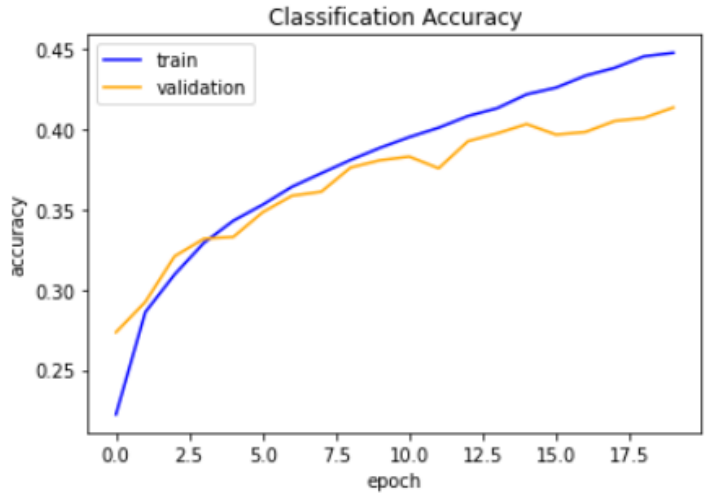
Training process :

```
Epoch 1/20
625/625 [==============================] - 5s 5ms/step - loss: 2.1330 - accuracy: 0.2222 - val_loss: 2.0390 - val_accuracy: 0.2734
Epoch 2/20
625/625 [==============================] - 3s 5ms/step - loss: 2.0029 - accuracy: 0.2860 - val_loss: 1.9756 - val_accuracy: 0.2923
Epoch 3/20
625/625 [==============================] - 3s 5ms/step - loss: 1.9446 - accuracy: 0.3095 - val_loss: 1.9380 - val_accuracy: 0.3208
Epoch 4/20
625/625 [==============================] - 3s 5ms/step - loss: 1.8992 - accuracy: 0.3292 - val_loss: 1.8974 - val_accuracy: 0.3317
Epoch 5/20
625/625 [==============================] - 3s 5ms/step - loss: 1.8620 - accuracy: 0.3429 - val_loss: 1.8764 - val_accuracy: 0.3328
Epoch 6/20
625/625 [==============================] - 3s 5ms/step - loss: 1.8326 - accuracy: 0.3529 - val_loss: 1.8459 - val_accuracy: 0.3482
Epoch 7/20
625/625 [==============================] - 3s 5ms/step - loss: 1.8058 - accuracy: 0.3641 - val_loss: 1.8079 - val_accuracy: 0.3585
Epoch 8/20
625/625 [==============================] - 3s 5ms/step - loss: 1.7796 - accuracy: 0.3726 - val_loss: 1.8033 - val_accuracy: 0.3610
Epoch 9/20
625/625 [==============================] - 3s 5ms/step - loss: 1.7584 - accuracy: 0.3809 - val_loss: 1.7736 - val_accuracy: 0.3761
Epoch 10/20
625/625 [==============================] - 3s 5ms/step - loss: 1.7350 - accuracy: 0.3884 - val_loss: 1.7615 - val_accuracy: 0.3806
Epoch 11/20
625/625 [==============================] - 3s 5ms/step - loss: 1.7151 - accuracy: 0.3951 - val_loss: 1.7506 - val_accuracy: 0.3829
Epoch 12/20
625/625 [==============================] - 3s 5ms/step - loss: 1.6946 - accuracy: 0.4010 - val_loss: 1.7451 - val_accuracy: 0.3756
Epoch 13/20

Epoch 13/20
625/625 [==============================] - 3s 5ms/step - loss: 1.6786 - accuracy: 0.4082 - val_loss: 1.7259 - val_accuracy: 0.3925
Epoch 14/20
625/625 [==============================] - 3s 5ms/step - loss: 1.6630 - accuracy: 0.4132 - val_loss: 1.7063 - val_accuracy: 0.3974
Epoch 15/20
625/625 [==============================] - 3s 5ms/step - loss: 1.6443 - accuracy: 0.4218 - val_loss: 1.6957 - val_accuracy: 0.4032
Epoch 16/20
625/625 [==============================] - 3s 5ms/step - loss: 1.6255 - accuracy: 0.4259 - val_loss: 1.6929 - val_accuracy: 0.3966
Epoch 17/20
625/625 [==============================] - 3s 5ms/step - loss: 1.6111 - accuracy: 0.4334 - val_loss: 1.7003 - val_accuracy: 0.3982
Epoch 18/20
625/625 [==============================] - 3s 5ms/step - loss: 1.5955 - accuracy: 0.4383 - val_loss: 1.6787 - val_accuracy: 0.4051
Epoch 19/20
625/625 [==============================] - 3s 4ms/step - loss: 1.5797 - accuracy: 0.4455 - val_loss: 1.6775 - val_accuracy: 0.4070
Epoch 20/20
625/625 [==============================] - 3s 5ms/step - loss: 1.5675 - accuracy: 0.4476 - val_loss: 1.6554 - val_accuracy: 0.4135
313/313 [==============================] - 1s 4ms/step - loss: 1.6543 - accuracy: 0.4103
```

Results:






```
Training time: 82.66622257232666s
test loss:  1.6543265581130981
test acc:  41.029998660087585
f1 score:   0.40452794132310527
recall score:  0.4103
precision score:  0.41781449863864417
confusion matrix:

 [[520  30  94  20  39   9  51  29 160  48]
 [ 67 423   7  23  17   9  61  29 122 242]
 [147  16 343  95  66  43 168  50  49  23]
 [ 83  25 103 260  57 117 175  61  44  75]
 [174  20 186  43 244  36 144  75  51  27]
 [ 76  13 110 182  38 269 130  87  51  44]
 [ 84  34  90  61  56  21 523  44  40  47]
 [ 86  21  87  68  58  44  70 439  39  88]
 [169  61  28  25  21  13  23  34 555  71]
 [ 66 141  28  42  12   7  46  40  91 527]]
```

Number of hidden layer neurons : 256

Architecture of neural net :

In this part I have used **four layers** neural net which consist **two hidden layers**. At the final layer I have used **softmax** as of our activation-function and **ReLu** at the hidden layers.

I have used **512 neurons** in first layer , **256 neurons** in second layer and **256 neurons** in third layer and **10 neurons** at the final layer.

Results:

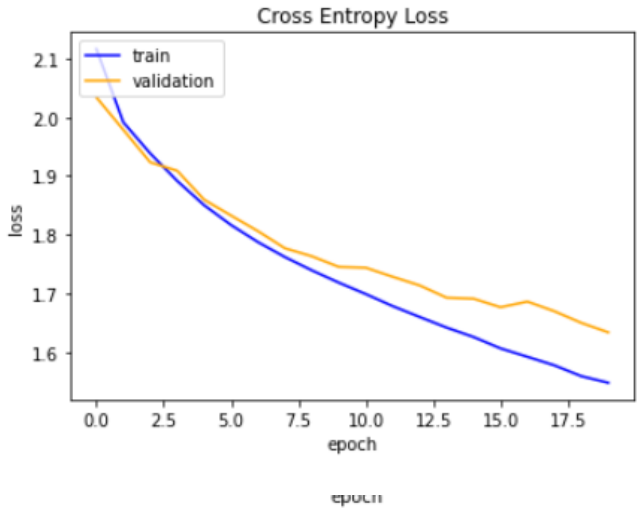
Training process :

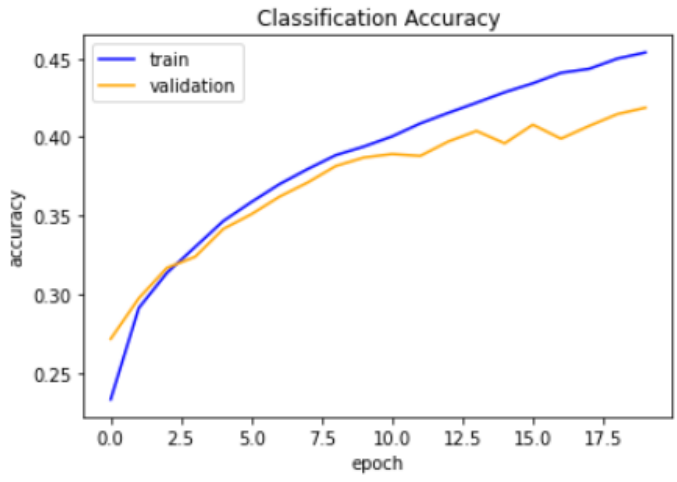
```
Epoch 1/20
625/625 [==============================] - 4s 5ms/step - loss: 2.1165 - accuracy: 0.2331 - val_loss: 2.0341 - val_accuracy: 0.2715
Epoch 2/20
625/625 [==============================] - 3s 5ms/step - loss: 1.9916 - accuracy: 0.2913 - val_loss: 1.9787 - val_accuracy: 0.2974
Epoch 3/20
625/625 [==============================] - 3s 5ms/step - loss: 1.9385 - accuracy: 0.3136 - val_loss: 1.9234 - val_accuracy: 0.3168
Epoch 4/20
625/625 [==============================] - 3s 5ms/step - loss: 1.8916 - accuracy: 0.3301 - val_loss: 1.9090 - val_accuracy: 0.3238
Epoch 5/20
625/625 [==============================] - 3s 5ms/step - loss: 1.8503 - accuracy: 0.3466 - val_loss: 1.8597 - val_accuracy: 0.3415
Epoch 6/20
625/625 [==============================] - 3s 5ms/step - loss: 1.8165 - accuracy: 0.3587 - val_loss: 1.8327 - val_accuracy: 0.3509
Epoch 7/20
625/625 [==============================] - 3s 5ms/step - loss: 1.7874 - accuracy: 0.3701 - val_loss: 1.8060 - val_accuracy: 0.3621
Epoch 8/20
625/625 [==============================] - 3s 5ms/step - loss: 1.7619 - accuracy: 0.3797 - val_loss: 1.7769 - val_accuracy: 0.3711
Epoch 9/20
625/625 [==============================] - 3s 5ms/step - loss: 1.7393 - accuracy: 0.3886 - val_loss: 1.7633 - val_accuracy: 0.3816
Epoch 10/20
625/625 [==============================] - 3s 5ms/step - loss: 1.7184 - accuracy: 0.3940 - val_loss: 1.7452 - val_accuracy: 0.3870
Epoch 11/20
625/625 [==============================] - 3s 5ms/step - loss: 1.6989 - accuracy: 0.4005 - val_loss: 1.7438 - val_accuracy: 0.3892
Epoch 12/20
625/625 [==============================] - 3s 5ms/step - loss: 1.6783 - accuracy: 0.4087 - val_loss: 1.7283 - val_accuracy: 0.3880
Epoch 13/20
625/625 [==============================] - 3s 5ms/step - loss: 1.6599 - accuracy: 0.4155 - val_loss: 1.7137 - val_accuracy: 0.3972
Epoch 13/20
625/625 [==============================] - 3s 5ms/step - loss: 1.6599 - accuracy: 0.4155 - val_loss: 1.7137 - val_accuracy: 0.3972
Epoch 14/20
625/625 [==============================] - 3s 5ms/step - loss: 1.6418 - accuracy: 0.4219 - val_loss: 1.6928 - val_accuracy: 0.4039
Epoch 15/20
625/625 [==============================] - 3s 5ms/step - loss: 1.6258 - accuracy: 0.4286 - val_loss: 1.6911 - val_accuracy: 0.3960
Epoch 16/20
625/625 [==============================] - 3s 5ms/step - loss: 1.6064 - accuracy: 0.4342 - val_loss: 1.6765 - val_accuracy: 0.4079
Epoch 17/20
625/625 [==============================] - 3s 5ms/step - loss: 1.5920 - accuracy: 0.4410 - val_loss: 1.6861 - val_accuracy: 0.3990
Epoch 18/20
625/625 [==============================] - 3s 5ms/step - loss: 1.5775 - accuracy: 0.4435 - val_loss: 1.6697 - val_accuracy: 0.4071
Epoch 19/20
625/625 [==============================] - 3s 5ms/step - loss: 1.5590 - accuracy: 0.4500 - val_loss: 1.6499 - val_accuracy: 0.4146
Epoch 20/20
625/625 [==============================] - 3s 5ms/step - loss: 1.5478 - accuracy: 0.4539 - val_loss: 1.6337 - val_accuracy: 0.4187
313/313 [==============================] - 1s 4ms/step - loss: 1.6307 - accuracy: 0.4203
```

Results:

```
Training time: 61.77888631820679s
test loss:  1.6306750774383545
test acc:  42.03000068664551
f1 score:   0.41939911127021834
recall score:  0.4203
precision score:  0.4246673111441504
confusion matrix:

 [[360  38 170  17 117  23  46  50 132  47]
 [ 30 472  13  23  33  20  49  38  81 241]
 [ 44  17 418  77 131  77 118  58  35  25]
 [ 32  28 128 231  75 180 144  83  30  69]
 [ 50  28 206  44 357  60 104  88  34  29]
 [ 16  14 155 133  69 392  70  85  32  34]
 [ 20  36 113  69 132  65 450  55  28  32]
 [ 36  31 101  61  77  90  32 477  28  67]
 [127  81  44  25  47  35  28  38 512  63]
 [ 36 145  35  32  22  25  44  62  65 534]]
```





Number of hidden layer neurons : 512

Architecture of neural net :

In this part I have used **four layers** neural net which consist **two hidden layers**. At the final layer I have used **softmax** as of our activation-function and **ReLu** at the hidden layers.

I have used **512 neurons** in first layer , **512 neurons** in second layer and **512 neurons** in third layer and **10 neurons** at the final layer.

Results:

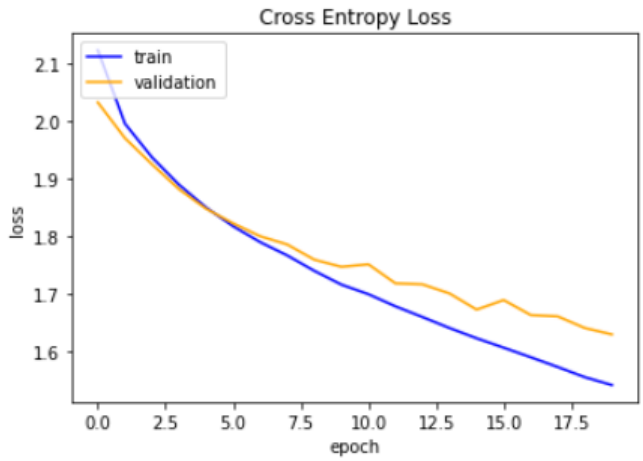
Training process :

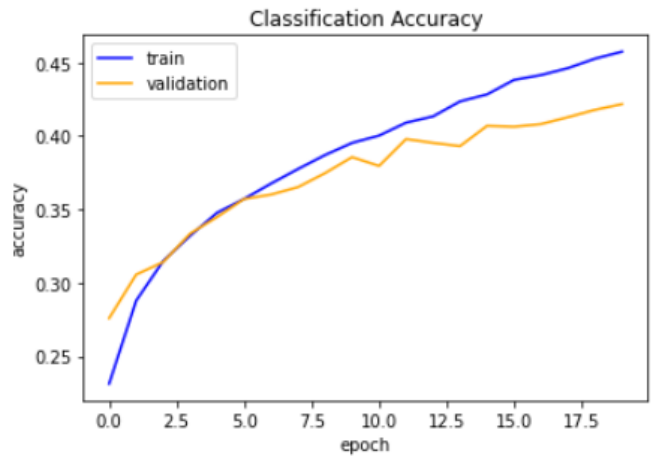
```
Epoch 1/20
625/625 [==============================] - 4s 5ms/step - loss: 2.1222 - accuracy: 0.2313 - val_loss: 2.0318 - val_accuracy: 0.2759
Epoch 2/20
625/625 [==============================] - 3s 5ms/step - loss: 1.9952 - accuracy: 0.2878 - val_loss: 1.9700 - val_accuracy: 0.3056
Epoch 3/20
625/625 [==============================] - 3s 5ms/step - loss: 1.9369 - accuracy: 0.3148 - val_loss: 1.9248 - val_accuracy: 0.3141
Epoch 4/20
625/625 [==============================] - 3s 5ms/step - loss: 1.8895 - accuracy: 0.3321 - val_loss: 1.8816 - val_accuracy: 0.3335
Epoch 5/20
625/625 [==============================] - 3s 5ms/step - loss: 1.8500 - accuracy: 0.3478 - val_loss: 1.8479 - val_accuracy: 0.3448
Epoch 6/20
625/625 [==============================] - 3s 5ms/step - loss: 1.8175 - accuracy: 0.3572 - val_loss: 1.8226 - val_accuracy: 0.3571
Epoch 7/20
625/625 [==============================] - 3s 5ms/step - loss: 1.7901 - accuracy: 0.3677 - val_loss: 1.8003 - val_accuracy: 0.3601
Epoch 8/20
625/625 [==============================] - 3s 5ms/step - loss: 1.7671 - accuracy: 0.3776 - val_loss: 1.7860 - val_accuracy: 0.3652
Epoch 9/20
625/625 [==============================] - 3s 5ms/step - loss: 1.7404 - accuracy: 0.3871 - val_loss: 1.7598 - val_accuracy: 0.3747
Epoch 10/20
625/625 [==============================] - 3s 5ms/step - loss: 1.7166 - accuracy: 0.3953 - val_loss: 1.7474 - val_accuracy: 0.3856
Epoch 11/20
625/625 [==============================] - 3s 5ms/step - loss: 1.7000 - accuracy: 0.4003 - val_loss: 1.7517 - val_accuracy: 0.3796
Epoch 12/20
625/625 [==============================] - 3s 5ms/step - loss: 1.6790 - accuracy: 0.4091 - val_loss: 1.7187 - val_accuracy: 0.3979
Epoch 13/20
625/625 [==============================] - 3s 5ms/step - loss: 1.6605 - accuracy: 0.4134 - val_loss: 1.7170 - val_accuracy: 0.3953
Epoch 14/20
Epoch 15/20
625/625 [==============================] - 3s 5ms/step - loss: 1.6237 - accuracy: 0.4284 - val_loss: 1.6733 - val_accuracy: 0.4069
Epoch 16/20
625/625 [==============================] - 3s 5ms/step - loss: 1.6075 - accuracy: 0.4383 - val_loss: 1.6899 - val_accuracy: 0.4063
Epoch 17/20
625/625 [==============================] - 3s 5ms/step - loss: 1.5909 - accuracy: 0.4415 - val_loss: 1.6640 - val_accuracy: 0.4081
Epoch 18/20
625/625 [==============================] - 3s 5ms/step - loss: 1.5740 - accuracy: 0.4462 - val_loss: 1.6616 - val_accuracy: 0.4128
Epoch 19/20
625/625 [==============================] - 3s 5ms/step - loss: 1.5564 - accuracy: 0.4527 - val_loss: 1.6415 - val_accuracy: 0.4178
Epoch 20/20
625/625 [==============================] - 3s 5ms/step - loss: 1.5428 - accuracy: 0.4575 - val_loss: 1.6304 - val_accuracy: 0.4217
313/313 [==============================] - 1s 4ms/step - loss: 1.6347 - accuracy: 0.4205
```

Results:

```
Training time: 64.87827897071838s
test loss:  1.6347222328186035
test acc:  42.05000102519989
f1 score:   0.414754818698287
recall score:  0.4205
precision score:  0.4246855199072821
confusion matrix:

 [[498  52  59  15 151  16  43  36  85  45]
 [ 55 571   6  18  33  12  38  40  55 172]
 [ 95  31 255  63 260  74  91  79  30  22]
 [ 50  55  56 193 147 156 158  81  29  75]
 [101  37  83  27 495  40  85  88  24  20]
 [ 48  27  66 130 123 337  83 112  32  42]
 [ 54  60  52  50 193  51 425  59  16  40]
 [ 66  43  42  51 131  48  38 484  23  74]
 [215 112  16  18  61  27  20  37 430  64]
 [ 47 221  20  26  24  13  31  52  49 517]]
```





As you can see the best neuron number is **512 for hidden layers.**

2.1.4 BATCH SIZE ANALYSE:

In this part we have chosen different numbers of 32,64,128,512 as of our batch-size

Batch size : 32

Architecture of neural net :

In this part I have used **four layers** neural net which consist **two hidden layers**. At the final layer I have used **softmax** as of our activation-function and **ReLu** at the hidden layers.

I have used **512 neurons** in first layer , **512 neurons** in second layer and **512 neurons** in third layer and **10 neurons** at the final layer.

Results:

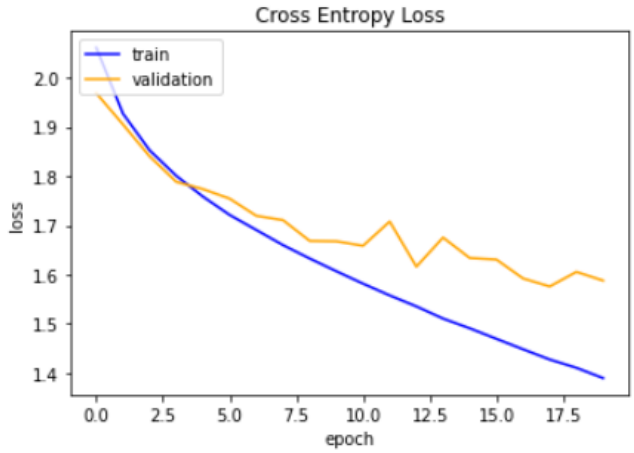
Training process :

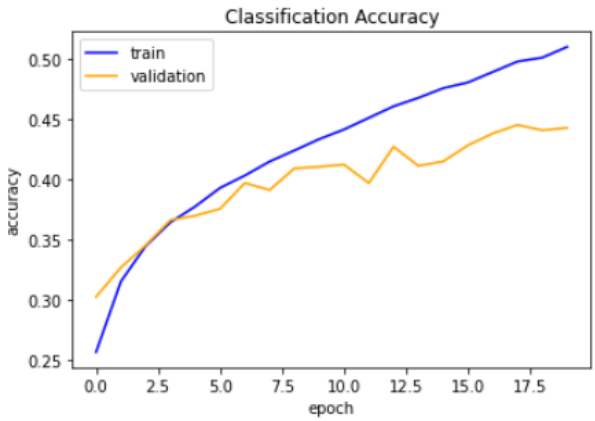
```
Epoch 1/20
1250/1250 [==============================] - 7s 5ms/step - loss: 2.0609 - accuracy: 0.2566 - val_loss: 1.9685 - val_accuracy: 0.3025
Epoch 2/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.9270 - accuracy: 0.3153 - val_loss: 1.9050 - val_accuracy: 0.3269
Epoch 3/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.8525 - accuracy: 0.3448 - val_loss: 1.8401 - val_accuracy: 0.3453
Epoch 4/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.8007 - accuracy: 0.3645 - val_loss: 1.7884 - val_accuracy: 0.3664
Epoch 5/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.7584 - accuracy: 0.3777 - val_loss: 1.7737 - val_accuracy: 0.3700
Epoch 6/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.7212 - accuracy: 0.3931 - val_loss: 1.7543 - val_accuracy: 0.3757
Epoch 7/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6906 - accuracy: 0.4034 - val_loss: 1.7194 - val_accuracy: 0.3972
Epoch 8/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6599 - accuracy: 0.4151 - val_loss: 1.7106 - val_accuracy: 0.3913
Epoch 9/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6327 - accuracy: 0.4243 - val_loss: 1.6685 - val_accuracy: 0.4094
Epoch 10/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6068 - accuracy: 0.4337 - val_loss: 1.6677 - val_accuracy: 0.4107
Epoch 11/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5816 - accuracy: 0.4417 - val_loss: 1.6585 - val_accuracy: 0.4125
Epoch 12/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5577 - accuracy: 0.4514 - val_loss: 1.7079 - val_accuracy: 0.3971
Epoch 13/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5354 - accuracy: 0.4610 - val_loss: 1.6161 - val_accuracy: 0.4273
Epoch 14/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5106 - accuracy: 0.4680 - val_loss: 1.6753 - val_accuracy: 0.4114
Epoch 15/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4909 - accuracy: 0.4760 - val_loss: 1.6337 - val_accuracy: 0.4151
Epoch 16/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4695 - accuracy: 0.4809 - val_loss: 1.6306 - val_accuracy: 0.4285
Epoch 17/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4483 - accuracy: 0.4895 - val_loss: 1.5922 - val_accuracy: 0.4384
Epoch 18/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4274 - accuracy: 0.4983 - val_loss: 1.5759 - val_accuracy: 0.4455
Epoch 19/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4106 - accuracy: 0.5015 - val_loss: 1.6055 - val_accuracy: 0.4410
Epoch 20/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.3897 - accuracy: 0.5104 - val_loss: 1.5879 - val_accuracy: 0.4431
313/313 [==============================] - 1s 4ms/step - loss: 1.5870 - accuracy: 0.4356
```

Results:

```
Training time: 142.44206476211548s
test loss:  1.5869795083999634
test acc:  43.560001254081726
f1 score:   0.4323836761938589
recall score:  0.4356
precision score:  0.4466294224185309
confusion matrix:

 [[407  48 155  14 116  11  27  65 112  45]
 [ 39 566  14  15  37  10  36  73  73 137]
 [ 48  21 448  52 159  56  64 110  27  15]
 [ 32  38 159 220 100 133 102 128  26  62]
 [ 63  17 213  30 399  33  64 126  37  18]
 [ 24  12 149 129  87 310  50 187  25  27]
 [ 28  49 134  67 144  42 388  90  24  34]
 [ 24  18  87  39  82  37  16 632  20  45]
 [132  86  39  16  63  19  11  66 510  58]
 [ 39 224  38  30  24   9  17  87  56 476]]
```





Batch size : 64

Architecture of neural net :

In this part I have used **four layers** neural net which consist **two hidden layers**. At the final layer I have used **softmax** as of our activation-function and **ReLu** at the hidden layers.

I have used **512 neurons** in first layer , **512 neurons** in second layer and **512 neurons** in third layer and **10 neurons** at the final layer.

Results:

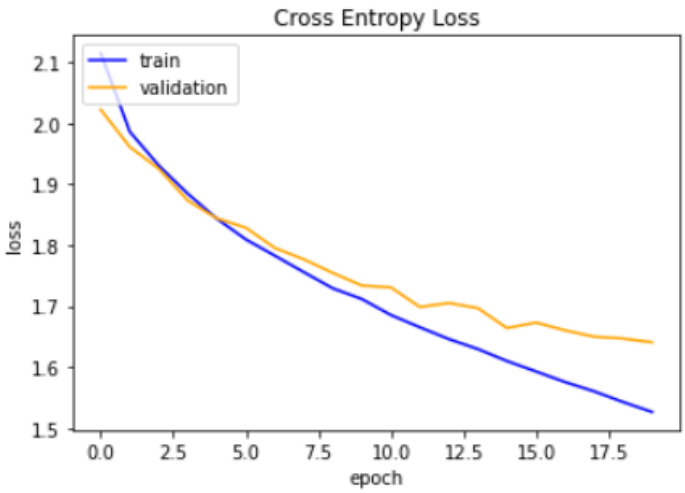
Training process :

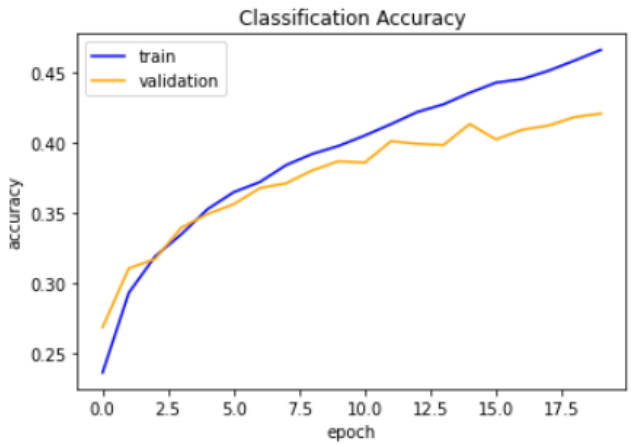
```
Epoch 1/20
625/625 [==============================] - 4s 6ms/step - loss: 2.1145 - accuracy: 0.2363 - val_loss: 2.0220 - val_accuracy: 0.2685
Epoch 2/20
625/625 [==============================] - 3s 5ms/step - loss: 1.9859 - accuracy: 0.2931 - val_loss: 1.9610 - val_accuracy: 0.3105
Epoch 3/20
625/625 [==============================] - 3s 5ms/step - loss: 1.9305 - accuracy: 0.3189 - val_loss: 1.9252 - val_accuracy: 0.3171
Epoch 4/20
625/625 [==============================] - 3s 5ms/step - loss: 1.8843 - accuracy: 0.3347 - val_loss: 1.8731 - val_accuracy: 0.3394
Epoch 5/20
625/625 [==============================] - 3s 5ms/step - loss: 1.8434 - accuracy: 0.3526 - val_loss: 1.8439 - val_accuracy: 0.3491
Epoch 6/20
625/625 [==============================] - 3s 5ms/step - loss: 1.8095 - accuracy: 0.3647 - val_loss: 1.8286 - val_accuracy: 0.3561
Epoch 7/20
625/625 [==============================] - 3s 5ms/step - loss: 1.7827 - accuracy: 0.3720 - val_loss: 1.7954 - val_accuracy: 0.3676
Epoch 8/20
625/625 [==============================] - 3s 5ms/step - loss: 1.7558 - accuracy: 0.3840 - val_loss: 1.7768 - val_accuracy: 0.3710
Epoch 9/20
625/625 [==============================] - 3s 5ms/step - loss: 1.7292 - accuracy: 0.3920 - val_loss: 1.7545 - val_accuracy: 0.3802
Epoch 10/20
625/625 [==============================] - 3s 5ms/step - loss: 1.7117 - accuracy: 0.3976 - val_loss: 1.7338 - val_accuracy: 0.3867
Epoch 11/20
625/625 [==============================] - 3s 5ms/step - loss: 1.6855 - accuracy: 0.4050 - val_loss: 1.7308 - val_accuracy: 0.3857
Epoch 12/20
625/625 [==============================] - 3s 5ms/step - loss: 1.6654 - accuracy: 0.4132 - val_loss: 1.6987 - val_accuracy: 0.4010
Epoch 13/20
625/625 [==============================] - 3s 5ms/step - loss: 1.6461 - accuracy: 0.4218 - val_loss: 1.7052 - val_accuracy: 0.3991
Epoch 14/20
625/625 [==============================] - 3s 5ms/step - loss: 1.6297 - accuracy: 0.4272 - val_loss: 1.6968 - val_accuracy: 0.3982
Epoch 15/20
625/625 [==============================] - 3s 5ms/step - loss: 1.6102 - accuracy: 0.4354 - val_loss: 1.6642 - val_accuracy: 0.4132
Epoch 16/20
625/625 [==============================] - 3s 5ms/step - loss: 1.5931 - accuracy: 0.4426 - val_loss: 1.6732 - val_accuracy: 0.4022
Epoch 17/20
625/625 [==============================] - 3s 5ms/step - loss: 1.5757 - accuracy: 0.4452 - val_loss: 1.6604 - val_accuracy: 0.4091
Epoch 18/20
625/625 [==============================] - 3s 5ms/step - loss: 1.5606 - accuracy: 0.4511 - val_loss: 1.6501 - val_accuracy: 0.4121
Epoch 19/20
625/625 [==============================] - 3s 5ms/step - loss: 1.5430 - accuracy: 0.4584 - val_loss: 1.6470 - val_accuracy: 0.4181
Epoch 20/20
625/625 [==============================] - 3s 5ms/step - loss: 1.5267 - accuracy: 0.4660 - val_loss: 1.6408 - val_accuracy: 0.4206
313/313 [==============================] - 1s 4ms/step - loss: 1.6411 - accuracy: 0.4169
```

Results:

```
Training time: 82.43347239494324s
test loss:  1.6411325931549072
test acc:  41.690000891685486
f1 score:   0.4133103398538572
recall score:  0.4169
precision score:  0.4252793789946456
confusion matrix:

 [[472  28 107  17  67  19  65  48 165  12]
 [ 66 492   9  32  24  17  63  43 157  97]
 [103  21 345  94 102  70 158  51  47   9]
 [ 53  29  99 253  68 153 174  87  53  31]
 [106  20 183  47 293  30 166  89  59   7]
 [ 47  16 108 160  51 342 124  88  52  12]
 [ 48  32  80  67  80  48 545  37  52  11]
 [ 86  26  69  74  75  62  61 488  35  24]
 [156  56  36  20  35  22  29  35 587  24]
 [ 66 201  24  48  22  14  70  80 123 352]]
```





Batch size : 128

Architecture of neural net :

In this part I have used **four layers** neural net which consist **two hidden layers**. At the final layer I have used **softmax** as of our activation-function and **ReLu** at the hidden layers.

I have used **512 neurons** in first layer , **512 neurons** in second layer and **512 neurons** in third layer and **10 neurons** at the final layer.

Results:

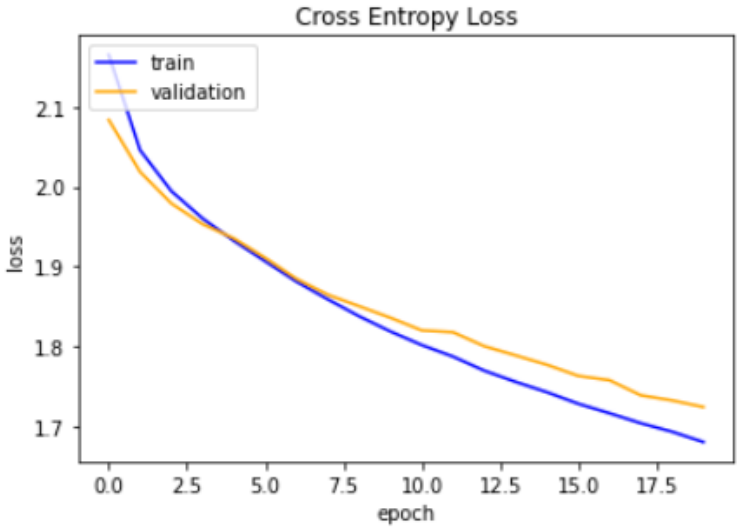
Training process :

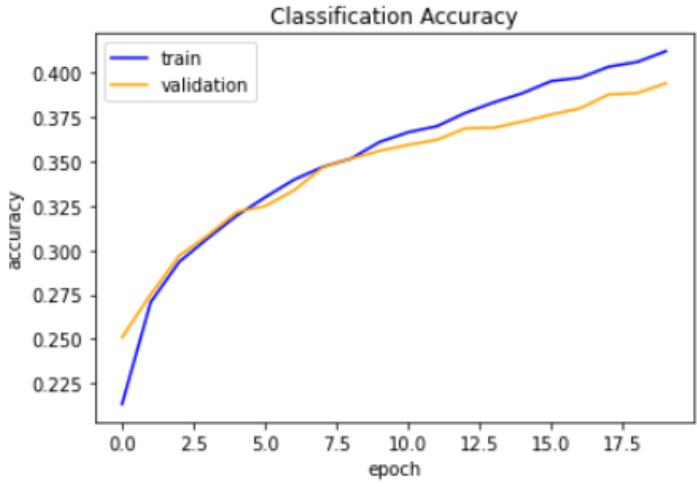
```
Epoch 1/20
313/313 [==============================] - 3s 7ms/step - loss: 2.1653 - accuracy: 0.2134 - val_loss: 2.0838 - val_accuracy: 0.2509
Epoch 2/20
313/313 [==============================] - 2s 6ms/step - loss: 2.0459 - accuracy: 0.2707 - val_loss: 2.0188 - val_accuracy: 0.2751
Epoch 3/20
313/313 [==============================] - 2s 6ms/step - loss: 1.9947 - accuracy: 0.2934 - val_loss: 1.9792 - val_accuracy: 0.2969
Epoch 4/20
313/313 [==============================] - 2s 6ms/step - loss: 1.9604 - accuracy: 0.3067 - val_loss: 1.9534 - val_accuracy: 0.3083
Epoch 5/20
313/313 [==============================] - 2s 6ms/step - loss: 1.9324 - accuracy: 0.3193 - val_loss: 1.9355 - val_accuracy: 0.3214
Epoch 6/20
313/313 [==============================] - 2s 6ms/step - loss: 1.9068 - accuracy: 0.3298 - val_loss: 1.9111 - val_accuracy: 0.3247
Epoch 7/20
313/313 [==============================] - 2s 6ms/step - loss: 1.8813 - accuracy: 0.3395 - val_loss: 1.8848 - val_accuracy: 0.3335
Epoch 8/20
313/313 [==============================] - 2s 6ms/step - loss: 1.8592 - accuracy: 0.3467 - val_loss: 1.8650 - val_accuracy: 0.3464
Epoch 9/20
313/313 [==============================] - 2s 6ms/step - loss: 1.8381 - accuracy: 0.3515 - val_loss: 1.8507 - val_accuracy: 0.3514
Epoch 10/20
313/313 [==============================] - 2s 6ms/step - loss: 1.8190 - accuracy: 0.3609 - val_loss: 1.8362 - val_accuracy: 0.3558
Epoch 11/20
313/313 [==============================] - 2s 6ms/step - loss: 1.8021 - accuracy: 0.3663 - val_loss: 1.8204 - val_accuracy: 0.3592
Epoch 12/20
313/313 [==============================] - 2s 6ms/step - loss: 1.7876 - accuracy: 0.3697 - val_loss: 1.8181 - val_accuracy: 0.3620
Epoch 13/20
313/313 [==============================] - 2s 6ms/step - loss: 1.7699 - accuracy: 0.3772 - val_loss: 1.8006 - val_accuracy: 0.3685
Epoch 14/20
313/313 [==============================] - 2s 6ms/step - loss: 1.7559 - accuracy: 0.3832 - val_loss: 1.7891 - val_accuracy: 0.3688
Epoch 15/20
313/313 [==============================] - 2s 6ms/step - loss: 1.7431 - accuracy: 0.3883 - val_loss: 1.7774 - val_accuracy: 0.3724
 Epoch 15/20
 313/313 [==============================] - 2s 6ms/step - loss: 1.7431 - accuracy: 0.3883 - val_loss: 1.7774 - val_accuracy: 0.3724
 Epoch 16/20
 313/313 [==============================] - 2s 6ms/step - loss: 1.7286 - accuracy: 0.3951 - val_loss: 1.7635 - val_accuracy: 0.3762
 Epoch 17/20
 313/313 [==============================] - 2s 6ms/step - loss: 1.7167 - accuracy: 0.3970 - val_loss: 1.7581 - val_accuracy: 0.3797
 Epoch 18/20
 313/313 [==============================] - 2s 6ms/step - loss: 1.7042 - accuracy: 0.4031 - val_loss: 1.7391 - val_accuracy: 0.3875
 Epoch 19/20
 313/313 [==============================] - 2s 6ms/step - loss: 1.6935 - accuracy: 0.4058 - val_loss: 1.7327 - val_accuracy: 0.3883
 Epoch 20/20
 313/313 [==============================] - 2s 6ms/step - loss: 1.6807 - accuracy: 0.4119 - val_loss: 1.7245 - val_accuracy: 0.3938
 313/313 [==============================] - 1s 4ms/step - loss: 1.7241 - accuracy: 0.3918
```

Results:

```
Training time: 41.50418567657471s
test loss:  1.7241413593292236
test acc:  39.17999863624573
f1 score:   0.3817189490545878
recall score:  0.3918
precision score:  0.387024939575022
confusion matrix:

 [[336  53  52  16  92  18  68  63 228  74]
 [ 12 491   6  21  22  15  42  50  97 244]
 [ 93  37 189  98 146  72 170 104  60  31]
 [ 46  36  45 206  94 160 156 109  57  91]
 [ 71  47  78  40 342  55 131 119  77  40]
 [ 40  30  56 156  81 303 110 113  55  56]
 [ 33  66  37  71 105  71 445  54  48  70]
 [ 35  49  34  60  65  62  59 480  61  95]
 [ 53 106  17  22  37  28  23  45 581  88]
 [ 17 172  12  29  14  17  41  55  98 545]]
```





Batch size : 512

Architecture of neural net :

In this part I have used **four layers** neural net which consist **two hidden layers**. At the final layer I have used **softmax** as of our activation-function and **ReLu** at the hidden layers.

I have used **512 neurons** in first layer , **512 neurons** in second layer and **512 neurons** in third layer and **10 neurons** at the final layer.

Results:

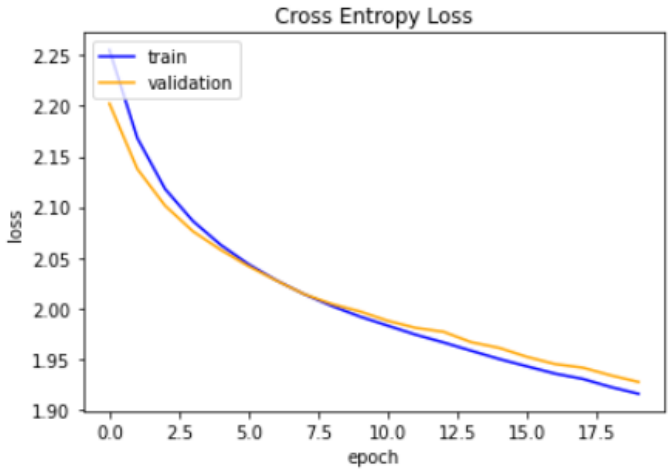
Training process :

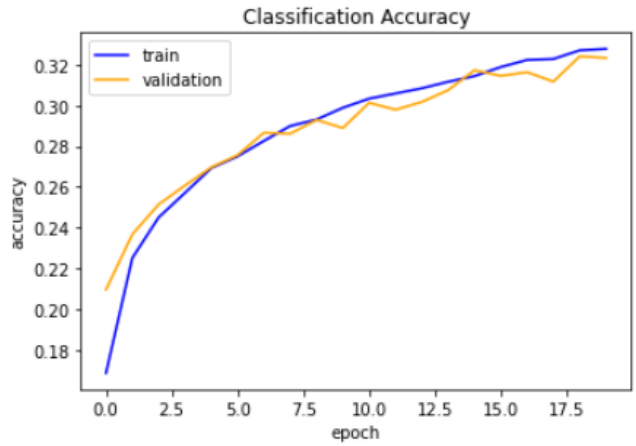
```
Epoch 1/20
79/79 [==============================] - 1s 11ms/step - loss: 2.2553 - accuracy: 0.1687 - val_loss: 2.2023 - val_accuracy: 0.2096
Epoch 2/20
79/79 [==============================] - 1s 10ms/step - loss: 2.1683 - accuracy: 0.2251 - val_loss: 2.1378 - val_accuracy: 0.2367
Epoch 3/20
79/79 [==============================] - 1s 9ms/step - loss: 2.1179 - accuracy: 0.2450 - val_loss: 2.1014 - val_accuracy: 0.2514
Epoch 4/20
79/79 [==============================] - 1s 9ms/step - loss: 2.0862 - accuracy: 0.2571 - val_loss: 2.0760 - val_accuracy: 0.2607
Epoch 5/20
79/79 [==============================] - 1s 10ms/step - loss: 2.0626 - accuracy: 0.2693 - val_loss: 2.0577 - val_accuracy: 0.2696
Epoch 6/20
79/79 [==============================] - 1s 9ms/step - loss: 2.0435 - accuracy: 0.2750 - val_loss: 2.0415 - val_accuracy: 0.2756
Epoch 7/20
79/79 [==============================] - 1s 10ms/step - loss: 2.0276 - accuracy: 0.2826 - val_loss: 2.0273 - val_accuracy: 0.2866
Epoch 8/20
79/79 [==============================] - 1s 9ms/step - loss: 2.0138 - accuracy: 0.2900 - val_loss: 2.0139 - val_accuracy: 0.2861
Epoch 9/20
79/79 [==============================] - 1s 9ms/step - loss: 2.0022 - accuracy: 0.2931 - val_loss: 2.0042 - val_accuracy: 0.2929
Epoch 10/20
79/79 [==============================] - 1s 9ms/step - loss: 1.9918 - accuracy: 0.2989 - val_loss: 1.9969 - val_accuracy: 0.2889
Epoch 11/20
79/79 [==============================] - 1s 10ms/step - loss: 1.9828 - accuracy: 0.3034 - val_loss: 1.9876 - val_accuracy: 0.3013
Epoch 12/20
79/79 [==============================] - 1s 10ms/step - loss: 1.9739 - accuracy: 0.3059 - val_loss: 1.9807 - val_accuracy: 0.2979
Epoch 13/20
79/79 [==============================] - 1s 9ms/step - loss: 1.9662 - accuracy: 0.3085 - val_loss: 1.9768 - val_accuracy: 0.3017
Epoch 14/20
79/79 [==============================] - 1s 9ms/step - loss: 1.9580 - accuracy: 0.3117 - val_loss: 1.9665 - val_accuracy: 0.3075
Epoch 14/20
79/79 [==============================] - 1s 9ms/step - loss: 1.9580 - accuracy: 0.3117 - val_loss: 1.9665 - val_accuracy: 0.3075
Epoch 15/20
79/79 [==============================] - 1s 10ms/step - loss: 1.9499 - accuracy: 0.3144 - val_loss: 1.9609 - val_accuracy: 0.3173
Epoch 16/20
79/79 [==============================] - 1s 10ms/step - loss: 1.9426 - accuracy: 0.3189 - val_loss: 1.9520 - val_accuracy: 0.3145
Epoch 17/20
79/79 [==============================] - 1s 9ms/step - loss: 1.9355 - accuracy: 0.3224 - val_loss: 1.9449 - val_accuracy: 0.3163
Epoch 18/20
79/79 [==============================] - 1s 10ms/step - loss: 1.9302 - accuracy: 0.3228 - val_loss: 1.9414 - val_accuracy: 0.3117
Epoch 19/20
79/79 [==============================] - 1s 10ms/step - loss: 1.9222 - accuracy: 0.3271 - val_loss: 1.9338 - val_accuracy: 0.3241
Epoch 20/20
79/79 [==============================] - 1s 10ms/step - loss: 1.9155 - accuracy: 0.3278 - val_loss: 1.9272 - val_accuracy: 0.3233
313/313 [==============================] - 1s 4ms/step - loss: 1.9240 - accuracy: 0.3251
```

Results:

```
Training time: 21.07959246635437s
test loss:  1.923996090888977
test acc:  32.510000467300415
f1 score:   0.31754737635127855
recall score:  0.3251
precision score:  0.3227094539358315
confusion matrix:

 [[347  59 103  14  76  24  62  67 195  53]
 [ 35 459   7  23  27  38  57  59 134 161]
 [177  58 232  50 101 104 109  66  79  24]
 [ 82  76  48 125 100 216 111 106  86  50]
 [146  45 115  43 252  99 107  77  98  18]
 [ 86  38  66  90  86 345  78  93  89  29]
 [102 108  57  40  90 155 268  56  92  32]
 [ 82  55  75  40 102  77  59 333 107  70]
 [107 117  20  15  24  79  32  40 489  77]
 [ 42 241  15  18  18  25  35  54 151 401]]
```





As we expect the best model is belongs to **batch-size = 32 as well as we suppose when we try smaller than bach-size value it tends to act pretty much better.**

2.1.5 ACTIVATION FUNCTION ANALYSE:

In this part we have chosen softmax for the last layer and then try different activation-function for the hidden layers.

Activation-Function for hidden layer : ReLU

Architecture of neural net :

In this part I have used **four layers** neural net which consist **two hidden layers**. At the final layer I have used **softmax** as of our activation-function, **Relu** as of **second** and **third** activation layer.

I have used **512 neurons** in first layer , **512 neurons** in second layer and **512 neurons** in third layer and **10 neurons** at the final layer.

Results:

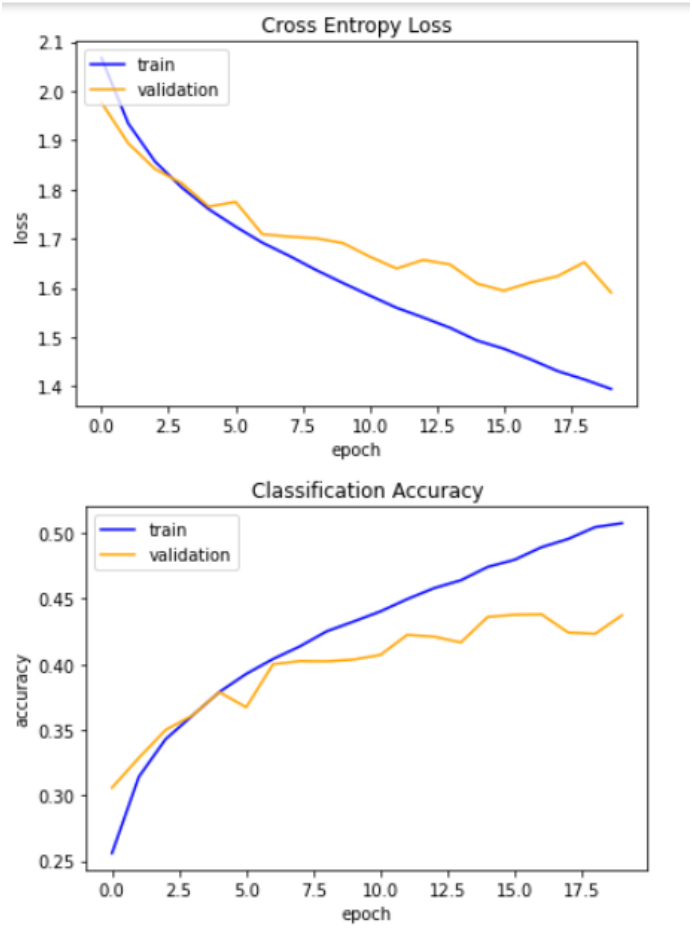
Training process :

```
Epoch 1/20
1250/1250 [==============================] - 7s 5ms/step - loss: 2.0669 - accuracy: 0.2560 - val_loss: 1.9764 - val_accuracy: 0.3058
Epoch 2/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.9337 - accuracy: 0.3142 - val_loss: 1.8933 - val_accuracy: 0.3286
Epoch 3/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.8563 - accuracy: 0.3429 - val_loss: 1.8413 - val_accuracy: 0.3499
Epoch 4/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.8034 - accuracy: 0.3609 - val_loss: 1.8122 - val_accuracy: 0.3609
Epoch 5/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.7596 - accuracy: 0.3788 - val_loss: 1.7650 - val_accuracy: 0.3787
Epoch 6/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.7243 - accuracy: 0.3926 - val_loss: 1.7745 - val_accuracy: 0.3671
Epoch 7/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6916 - accuracy: 0.4040 - val_loss: 1.7088 - val_accuracy: 0.3999
Epoch 8/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6649 - accuracy: 0.4135 - val_loss: 1.7039 - val_accuracy: 0.4023
Epoch 9/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6361 - accuracy: 0.4250 - val_loss: 1.7005 - val_accuracy: 0.4021
Epoch 10/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6098 - accuracy: 0.4325 - val_loss: 1.6910 - val_accuracy: 0.4035
Epoch 11/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5843 - accuracy: 0.4402 - val_loss: 1.6633 - val_accuracy: 0.4070
Epoch 12/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5594 - accuracy: 0.4496 - val_loss: 1.6392 - val_accuracy: 0.4223
Epoch 13/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5396 - accuracy: 0.4579 - val_loss: 1.6566 - val_accuracy: 0.4209
Epoch 14/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5186 - accuracy: 0.4640 - val_loss: 1.6473 - val_accuracy: 0.4165
Epoch 15/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4928 - accuracy: 0.4741 - val_loss: 1.6087 - val_accuracy: 0.4360
Epoch 16/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4762 - accuracy: 0.4796 - val_loss: 1.5940 - val_accuracy: 0.4376
Epoch 17/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4546 - accuracy: 0.4891 - val_loss: 1.6108 - val_accuracy: 0.4380
Epoch 18/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4309 - accuracy: 0.4954 - val_loss: 1.6234 - val_accuracy: 0.4241
Epoch 19/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4138 - accuracy: 0.5045 - val_loss: 1.6519 - val_accuracy: 0.4231
Epoch 20/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.3945 - accuracy: 0.5074 - val_loss: 1.5903 - val_accuracy: 0.4372
313/313 [==============================] - 1s 4ms/step - loss: 1.5955 - accuracy: 0.4354
```

Results:

```
Training time: 142.48153162002563s
test loss:  1.5954723358154297
test acc:  43.540000915527344
f1 score:   0.4256391200057179
recall score:  0.4354
precision score:  0.43955973940533294
confusion matrix:

 [[489  22  97  16  49  25  28  61 163  50]
 [ 31 357  13  18  16  22  40  54 119 330]
 [112  12 346  60  87 105  96 109  44  29]
 [ 61  16  96 195  54 227 119 103  26 103]
 [104   5 171  42 253  73 104 162  56  30]
 [ 44   5 105 108  28 425  63 150  34  38]
 [ 47  19  85  60  60  83 462  86  46  52]
 [ 40   8  62  46  25  69  25 624  27  74]
 [109  37  24  16  20  38  15  58 598  85]
 [ 47  77  21  30   9  24  23  69  95 605]]
```



Activation-Function for hidden layer : TanH

Architecture of neural net :

In this part I have used **four layers** neural net which consist **two hidden layers**. At the final layer I have used **softmax** as of our activation-function, **TanH** as of **second** and **third** activation layer.

I have used **512 neurons** in first layer , **512 neurons** in second layer and **512 neurons** in third layer and **10 neurons** at the final layer.

Results:

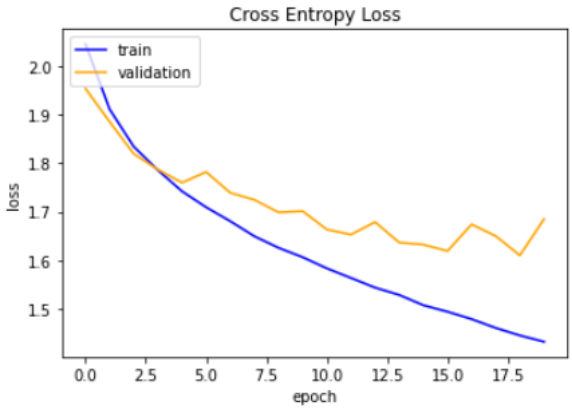
Training process :

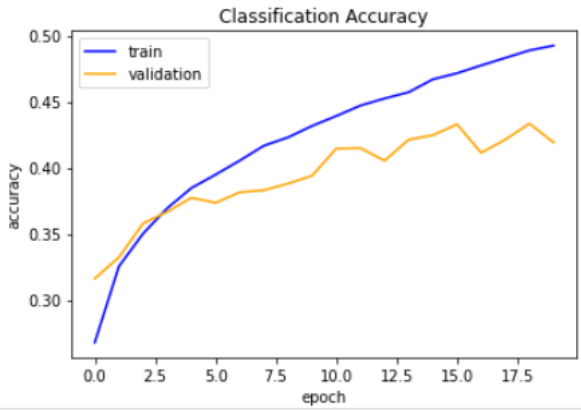
```
Epoch 1/20
1250/1250 [==============================] - 7s 5ms/step - loss: 2.0455 - accuracy: 0.2677 - val_loss: 1.9537 - val_accuracy: 0.3161
Epoch 2/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.9111 - accuracy: 0.3253 - val_loss: 1.8853 - val_accuracy: 0.3321
Epoch 3/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.8340 - accuracy: 0.3501 - val_loss: 1.8193 - val_accuracy: 0.3578
Epoch 4/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.7856 - accuracy: 0.3693 - val_loss: 1.7869 - val_accuracy: 0.3666
Epoch 5/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.7421 - accuracy: 0.3846 - val_loss: 1.7599 - val_accuracy: 0.3772
Epoch 6/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.7092 - accuracy: 0.3947 - val_loss: 1.7819 - val_accuracy: 0.3734
Epoch 7/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6804 - accuracy: 0.4053 - val_loss: 1.7390 - val_accuracy: 0.3813
Epoch 8/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6494 - accuracy: 0.4166 - val_loss: 1.7244 - val_accuracy: 0.3829
Epoch 9/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6257 - accuracy: 0.4229 - val_loss: 1.6987 - val_accuracy: 0.3879
Epoch 10/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6063 - accuracy: 0.4316 - val_loss: 1.7012 - val_accuracy: 0.3939
Epoch 11/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5834 - accuracy: 0.4391 - val_loss: 1.6634 - val_accuracy: 0.4144
Epoch 12/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5636 - accuracy: 0.4470 - val_loss: 1.6529 - val_accuracy: 0.4148
Epoch 13/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5437 - accuracy: 0.4523 - val_loss: 1.6787 - val_accuracy: 0.4052
Epoch 14/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5287 - accuracy: 0.4571 - val_loss: 1.6365 - val_accuracy: 0.4212
Epoch 15/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5074 - accuracy: 0.4669 - val_loss: 1.6323 - val_accuracy: 0.4245
Epoch 16/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4941 - accuracy: 0.4715 - val_loss: 1.6191 - val_accuracy: 0.4328
Epoch 17/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4789 - accuracy: 0.4774 - val_loss: 1.6740 - val_accuracy: 0.4112
Epoch 18/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4606 - accuracy: 0.4831 - val_loss: 1.6498 - val_accuracy: 0.4212
Epoch 19/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4452 - accuracy: 0.4888 - val_loss: 1.6101 - val_accuracy: 0.4333
Epoch 20/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4322 - accuracy: 0.4924 - val_loss: 1.6850 - val_accuracy: 0.4190
313/313 [==============================] - 1s 4ms/step - loss: 1.6809 - accuracy: 0.4161
```

Results:

```
Training time: 121.24407887458801s
test loss:  1.6808619499206543
test acc:  41.60999953746796
f1 score:  0.40129524226745567
recall score:  0.4161
precision score:  0.42395636144519094
confusion matrix:

[[510  43  31   8  49   5  18  23 284  29]
 [ 43 601   6   8  13  10  28  19 166 106]
 [226  28 198  55 128  71  92  60 123  19]
 [102  62  45 154  85 161 136  71 113  71]
 [187  46  68  16 311  36  88  81 142  25]
 [ 86  30  50  84  71 334  84 124  96  41]
 [108  83  38  37 100  34 418  27 121  34]
 [ 80  45  31  35  74  63  32 482 100  58]
 [ 91  72   4   6  15  18  14  16 734  30]
 [ 48 270  12   9  12  12  20  24 174 419]]
```





Activation-Function for hidden layer : Sigmoid

Architecture of neural net :

In this part I have used **four layers** neural net which consist **two hidden layers**. At the final layer I have used **softmax** as of our activation-function, **Sigmoid** as of **second** and **third** activation layer.

I have used **512 neurons** in first layer , **512 neurons** in second layer and **512 neurons** in third layer and **10 neurons** at the final layer.

Results:

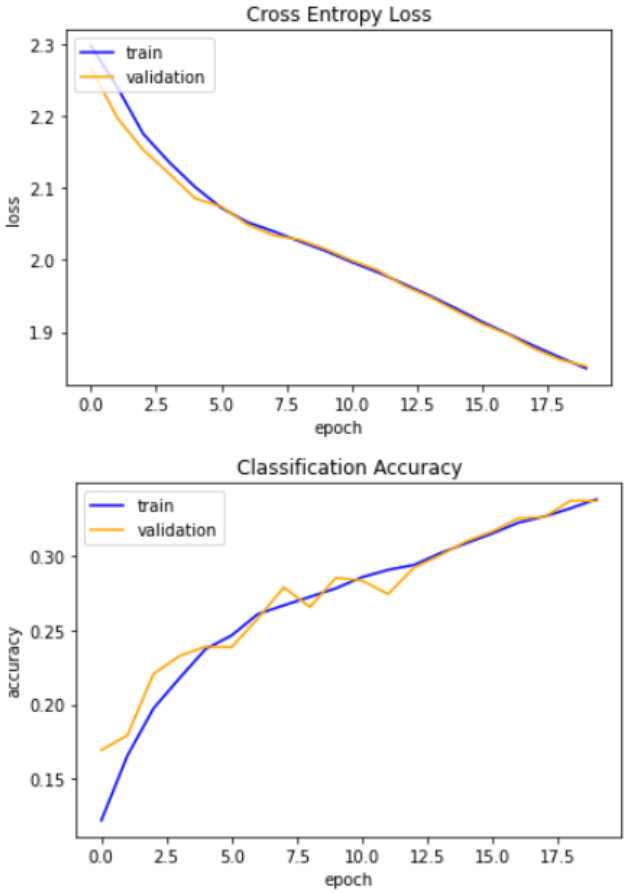
Training process :

```
Epoch 1/20
1250/1250 [==============================] - 6s 5ms/step - loss: 2.2967 - accuracy: 0.1220 - val_loss: 2.2666 - val_accuracy: 0.1693
Epoch 2/20
1250/1250 [==============================] - 6s 5ms/step - loss: 2.2401 - accuracy: 0.1656 - val_loss: 2.1976 - val_accuracy: 0.1792
Epoch 3/20
1250/1250 [==============================] - 6s 5ms/step - loss: 2.1751 - accuracy: 0.1973 - val_loss: 2.1528 - val_accuracy: 0.2205
Epoch 4/20
1250/1250 [==============================] - 6s 5ms/step - loss: 2.1357 - accuracy: 0.2177 - val_loss: 2.1198 - val_accuracy: 0.2327
Epoch 5/20
1250/1250 [==============================] - 6s 5ms/step - loss: 2.1010 - accuracy: 0.2373 - val_loss: 2.0853 - val_accuracy: 0.2389
Epoch 6/20
1250/1250 [==============================] - 6s 5ms/step - loss: 2.0723 - accuracy: 0.2465 - val_loss: 2.0743 - val_accuracy: 0.2385
Epoch 7/20
1250/1250 [==============================] - 6s 5ms/step - loss: 2.0525 - accuracy: 0.2608 - val_loss: 2.0492 - val_accuracy: 0.2578
Epoch 8/20
1250/1250 [==============================] - 6s 5ms/step - loss: 2.0397 - accuracy: 0.2668 - val_loss: 2.0342 - val_accuracy: 0.2788
Epoch 9/20
1250/1250 [==============================] - 6s 5ms/step - loss: 2.0259 - accuracy: 0.2724 - val_loss: 2.0280 - val_accuracy: 0.2656
Epoch 10/20
1250/1250 [==============================] - 7s 5ms/step - loss: 2.0125 - accuracy: 0.2782 - val_loss: 2.0151 - val_accuracy: 0.2852
Epoch 11/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.9970 - accuracy: 0.2857 - val_loss: 1.9993 - val_accuracy: 0.2834
Epoch 12/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.9830 - accuracy: 0.2906 - val_loss: 1.9859 - val_accuracy: 0.2743
Epoch 13/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.9668 - accuracy: 0.2939 - val_loss: 1.9645 - val_accuracy: 0.2922
Epoch 14/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.9504 - accuracy: 0.3018 - val_loss: 1.9486 - val_accuracy: 0.3004
1250/1250 [==============================] - 6s 5ms/step - loss: 1.9504 - accuracy: 0.3018 - val_loss: 1.9486 - val_accuracy: 0.3004
Epoch 15/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.9327 - accuracy: 0.3085 - val_loss: 1.9295 - val_accuracy: 0.3098
Epoch 16/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.9143 - accuracy: 0.3151 - val_loss: 1.9116 - val_accuracy: 0.3164
Epoch 17/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.8976 - accuracy: 0.3224 - val_loss: 1.8967 - val_accuracy: 0.3252
Epoch 18/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.8806 - accuracy: 0.3265 - val_loss: 1.8769 - val_accuracy: 0.3262
Epoch 19/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.8647 - accuracy: 0.3320 - val_loss: 1.8618 - val_accuracy: 0.3371
Epoch 20/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.8494 - accuracy: 0.3381 - val_loss: 1.8525 - val_accuracy: 0.3370
313/313 [==============================] - 1s 4ms/step - loss: 1.8450 - accuracy: 0.3432
```

Results:

```
Training time: 142.41431498527527s
test loss:  1.8450077772140503
test acc:  34.31999981403351
f1 score:   0.3248637086024922
recall score:  0.3432
precision score:  0.3375412689237408
confusion matrix:

 [[335  40  51  10 168  17  57  64 212  46]
 [ 29 418   6   0  44  20  80  42 157 204]
 [137  31 168  13 250  80 163  77  58  23]
 [ 71  46  42  25 151 226 171 131  78  59]
 [ 93  36  72   1 451  58 116  84  74  15]
 [ 70  25  55  21 126 338 123 121  91  30]
 [ 52  49  46  18 209 104 372  58  48  44]
 [ 60  40  41  11 170  71  75 353  93  86]
 [107  95  11   1  40  51  29  44 546  76]
 [ 31 198   8   9  40  13  58  58 159 426]]
```



As you can see the best case is belongs **to Relu activation function.**

2.1.6 LOSS FUNCTION ANALYSE:

In this part we have chosen softmax for the last layer and then try different loss function for the models

Loss Function: categorical_crossentropy

Architecture of neural net :

In this part I have used **four layers** neural net which consist **two hidden layers**. At the final layer I have used **softmax** as of our activation-function, **Relu** as of **second** and **third** activation layer.

I have used **512 neurons** in first layer , **512 neurons** in second layer and **512 neurons** in third layer and **10 neurons** at the final layer.

Results:

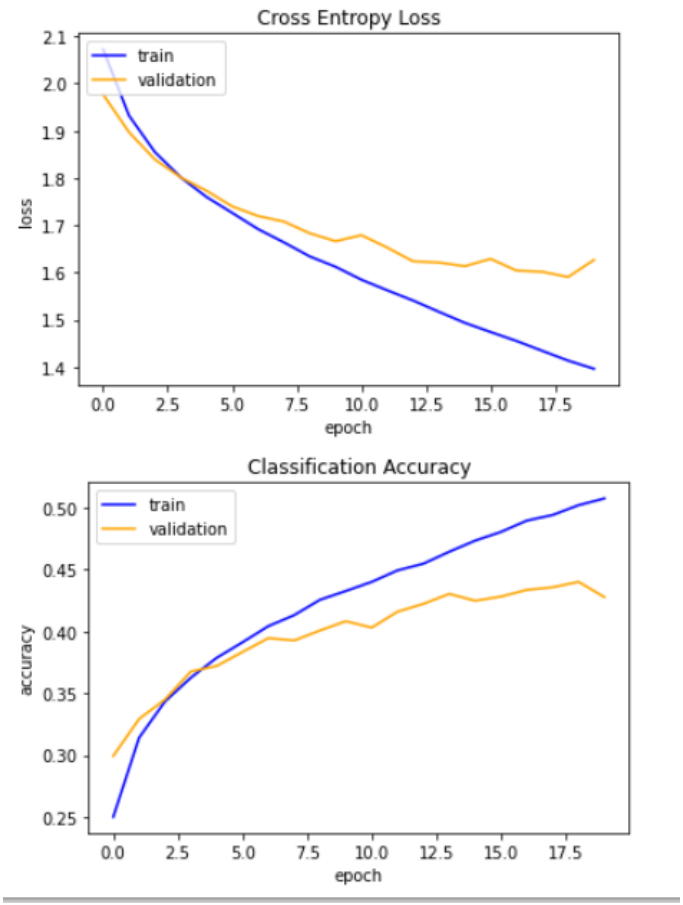
Training process :

```
Epoch 1/20
1250/1250 [==============================] - 7s 5ms/step - loss: 2.0724 - accuracy: 0.2501 - val_loss: 1.9769 - val_accuracy: 0.2994
Epoch 2/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.9324 - accuracy: 0.3142 - val_loss: 1.8972 - val_accuracy: 0.3293
Epoch 3/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.8553 - accuracy: 0.3434 - val_loss: 1.8394 - val_accuracy: 0.3450
Epoch 4/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.8017 - accuracy: 0.3626 - val_loss: 1.8015 - val_accuracy: 0.3676
Epoch 5/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.7599 - accuracy: 0.3787 - val_loss: 1.7731 - val_accuracy: 0.3721
Epoch 6/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.7262 - accuracy: 0.3913 - val_loss: 1.7399 - val_accuracy: 0.3834
Epoch 7/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6919 - accuracy: 0.4044 - val_loss: 1.7196 - val_accuracy: 0.3946
Epoch 8/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6638 - accuracy: 0.4132 - val_loss: 1.7078 - val_accuracy: 0.3927
Epoch 9/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6340 - accuracy: 0.4257 - val_loss: 1.6830 - val_accuracy: 0.4008
Epoch 10/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6121 - accuracy: 0.4327 - val_loss: 1.6664 - val_accuracy: 0.4083
Epoch 11/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5848 - accuracy: 0.4401 - val_loss: 1.6791 - val_accuracy: 0.4031
Epoch 12/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5623 - accuracy: 0.4494 - val_loss: 1.6527 - val_accuracy: 0.4160
Epoch 13/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5408 - accuracy: 0.4548 - val_loss: 1.6237 - val_accuracy: 0.4223
Epoch 14/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5169 - accuracy: 0.4645 - val_loss: 1.6211 - val_accuracy: 0.4304
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5169 - accuracy: 0.4645 - val_loss: 1.6211 - val_accuracy: 0.4304
Epoch 15/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4935 - accuracy: 0.4735 - val_loss: 1.6134 - val_accuracy: 0.4248
Epoch 16/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4741 - accuracy: 0.4805 - val_loss: 1.6289 - val_accuracy: 0.4282
Epoch 17/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4552 - accuracy: 0.4897 - val_loss: 1.6041 - val_accuracy: 0.4336
Epoch 18/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4343 - accuracy: 0.4942 - val_loss: 1.6016 - val_accuracy: 0.4357
Epoch 19/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4136 - accuracy: 0.5021 - val_loss: 1.5905 - val_accuracy: 0.4401
Epoch 20/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.3963 - accuracy: 0.5076 - val_loss: 1.6266 - val_accuracy: 0.4278
313/313 [==============================] - 1s 4ms/step - loss: 1.6233 - accuracy: 0.4281
```

Results:

```
Training time: 122.6294002532959s
test loss:  1.6232714653015137
test acc:  42.809998989105225
f1 score:   0.41596622059066457
recall score:  0.4281
precision score:  0.4507989787316822
confusion matrix:

 [[265  46  48  20 149   7  97  24 278  66]
 [  9 547   1  15  27   5  57  15 112 212]
 [ 32  33 196  73 228  64 215  42  83  34]
 [ 26  34  32 229 123 130 217  41  65 103]
 [ 30  28  45  30 434  32 217  50  94  40]
 [ 15  23  48 172  99 321 159  58  58  47]
 [ 14  53  21  44 112  23 628  16  45  44]
 [ 16  35  23  55 136  55 103 422  54 101]
 [ 37  78   7  19  47  14  34  10 672  82]
 [ 20 178   4  22  21  12  44  26 106 567]]
```



Loss Function: MSE

Architecture of neural net :

In this part I have used **four layers** neural net which consist **two hidden layers**. At the final layer I have used **softmax** as of our activation-function, **Relu** as of **second** and **third** activation layer.

I have used **512 neurons** in first layer , **512 neurons** in second layer and **512 neurons** in third layer and **10 neurons** at the final layer.

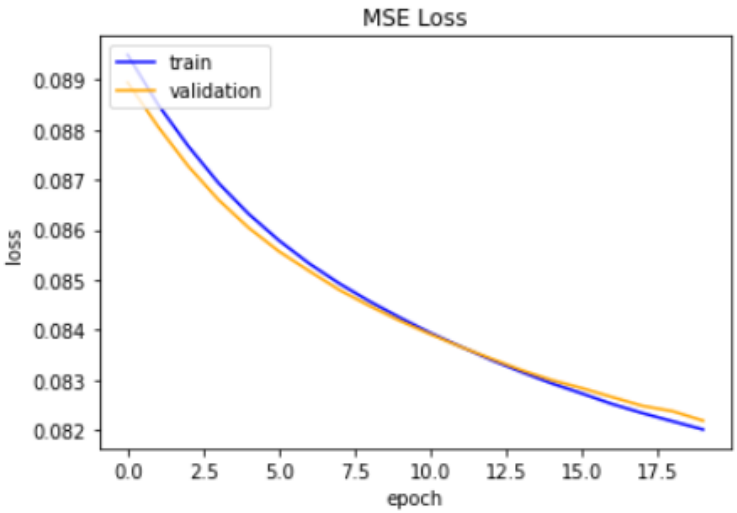
Results:

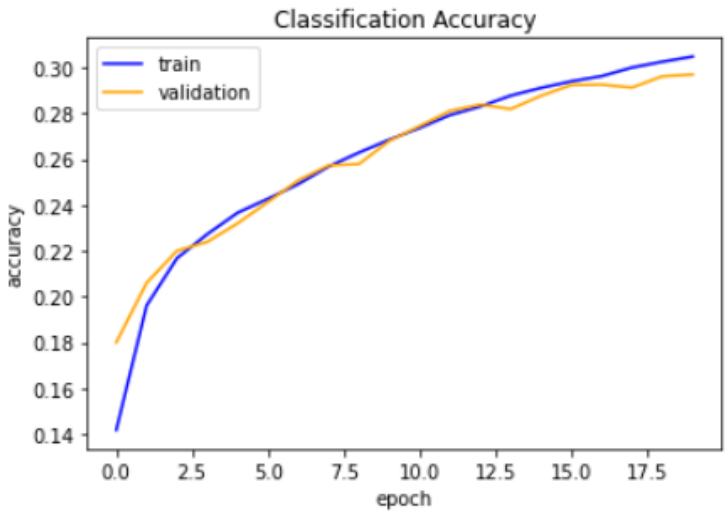
Training process :

```
Epoch 1/20
1250/1250 [==============================] - 7s 5ms/step - loss: 0.0895 - accuracy: 0.1417 - val_loss: 0.0889 - val_accuracy: 0.1800
Epoch 2/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.0885 - accuracy: 0.1961 - val_loss: 0.0881 - val_accuracy: 0.2060
Epoch 3/20
1250/1250 [==============================] - 6s 4ms/step - loss: 0.0877 - accuracy: 0.2168 - val_loss: 0.0873 - val_accuracy: 0.2200
Epoch 4/20
1250/1250 [==============================] - 6s 4ms/step - loss: 0.0869 - accuracy: 0.2273 - val_loss: 0.0866 - val_accuracy: 0.2239
Epoch 5/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.0863 - accuracy: 0.2367 - val_loss: 0.0860 - val_accuracy: 0.2320
Epoch 6/20
1250/1250 [==============================] - 6s 4ms/step - loss: 0.0858 - accuracy: 0.2428 - val_loss: 0.0856 - val_accuracy: 0.2413
Epoch 7/20
1250/1250 [==============================] - 6s 4ms/step - loss: 0.0853 - accuracy: 0.2491 - val_loss: 0.0852 - val_accuracy: 0.2508
Epoch 8/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.0849 - accuracy: 0.2569 - val_loss: 0.0848 - val_accuracy: 0.2573
Epoch 9/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.0846 - accuracy: 0.2629 - val_loss: 0.0845 - val_accuracy: 0.2579
Epoch 10/20
1250/1250 [==============================] - 5s 4ms/step - loss: 0.0842 - accuracy: 0.2686 - val_loss: 0.0842 - val_accuracy: 0.2680
Epoch 11/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.0839 - accuracy: 0.2737 - val_loss: 0.0839 - val_accuracy: 0.2745
Epoch 12/20
1250/1250 [==============================] - 6s 4ms/step - loss: 0.0837 - accuracy: 0.2793 - val_loss: 0.0837 - val_accuracy: 0.2811
Epoch 13/20
1250/1250 [==============================] - 6s 4ms/step - loss: 0.0834 - accuracy: 0.2831 - val_loss: 0.0834 - val_accuracy: 0.2839
Epoch 14/20
1250/1250 [==============================] - 5s 4ms/step - loss: 0.0832 - accuracy: 0.2878 - val_loss: 0.0832 - val_accuracy: 0.2819
Epoch 15/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.0829 - accuracy: 0.2912 - val_loss: 0.0830 - val_accuracy: 0.2877
Epoch 16/20
Epoch 16/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.0827 - accuracy: 0.2941 - val_loss: 0.0828 - val_accuracy: 0.2924
Epoch 17/20
1250/1250 [==============================] - 5s 4ms/step - loss: 0.0825 - accuracy: 0.2963 - val_loss: 0.0827 - val_accuracy: 0.2926
Epoch 18/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.0823 - accuracy: 0.3001 - val_loss: 0.0825 - val_accuracy: 0.2913
Epoch 19/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.0822 - accuracy: 0.3026 - val_loss: 0.0824 - val_accuracy: 0.2962
Epoch 20/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.0820 - accuracy: 0.3049 - val_loss: 0.0822 - val_accuracy: 0.2970
313/313 [==============================] - 1s 3ms/step - loss: 0.0821 - accuracy: 0.3026
```

```
Training time: 142.51552319526672s
test loss:  0.08210840821266174
test acc:  30.259999632835388
f1 score:   0.2886588864413675
recall score:  0.3026
precision score:  0.2977601246454793
confusion matrix:

 [[327  73 102   7  51  33  57  79 216  55]
 [ 51 407  16  17  33  59  31  57 144 185]
 [201  70 238  18  80 127  67  76 100  23]
 [ 79 104  88  58  78 258  57 128  90  60]
 [156  61 125  29 211 136  66  81 117  18]
 [ 82  56  89  48  71 369  51 120  84  30]
 [ 96 125  85  27  68 183 179  79 109  49]
 [ 84  76  75  21 105  84  42 316 110  87]
 [ 82 118  22  14  21  97  22  50 503  71]
 [ 41 214  26   9  15  34  27  66 150 418]]
```





Loss Function: MAE

Architecture of neural net :

In this part I have used **four layers** neural net which consist **two hidden layers**. At the final layer I have used **softmax** as of our activation-function, **Relu** as of **second** and **third** activation layer.

I have used **512 neurons** in first layer , **512 neurons** in second layer and **512 neurons** in third layer and **10 neurons** at the final layer.

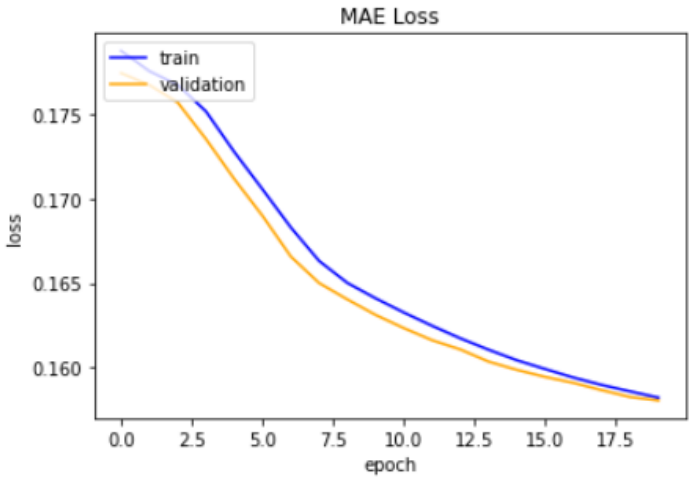
Results:

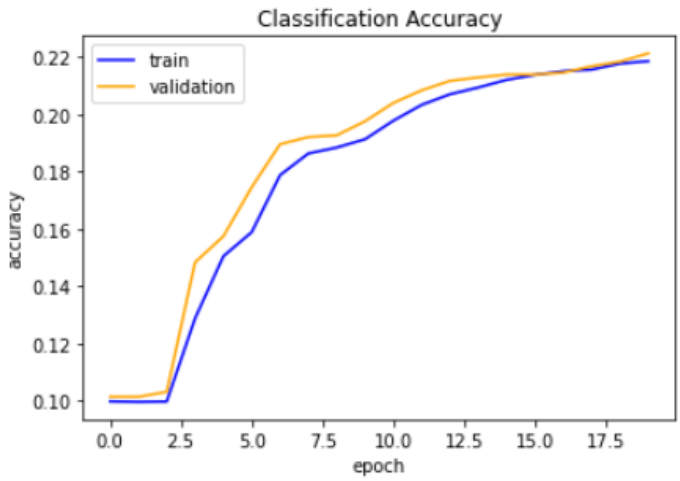
Training process :

```
Epoch 1/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.1787 - accuracy: 0.0998 - val_loss: 0.1774 - val_accuracy: 0.1014
Epoch 2/20
1250/1250 [==============================] - 5s 4ms/step - loss: 0.1775 - accuracy: 0.0997 - val_loss: 0.1767 - val_accuracy: 0.1014
Epoch 3/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.1767 - accuracy: 0.0997 - val_loss: 0.1757 - val_accuracy: 0.1031
Epoch 4/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.1752 - accuracy: 0.1288 - val_loss: 0.1735 - val_accuracy: 0.1483
Epoch 5/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.1728 - accuracy: 0.1505 - val_loss: 0.1712 - val_accuracy: 0.1574
Epoch 6/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.1705 - accuracy: 0.1588 - val_loss: 0.1690 - val_accuracy: 0.1745
Epoch 7/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.1683 - accuracy: 0.1787 - val_loss: 0.1666 - val_accuracy: 0.1895
Epoch 8/20
1250/1250 [==============================] - 5s 4ms/step - loss: 0.1663 - accuracy: 0.1863 - val_loss: 0.1650 - val_accuracy: 0.1920
Epoch 9/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.1650 - accuracy: 0.1884 - val_loss: 0.1640 - val_accuracy: 0.1926
Epoch 10/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.1641 - accuracy: 0.1912 - val_loss: 0.1631 - val_accuracy: 0.1975
Epoch 11/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.1633 - accuracy: 0.1977 - val_loss: 0.1623 - val_accuracy: 0.2039
Epoch 12/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.1625 - accuracy: 0.2033 - val_loss: 0.1616 - val_accuracy: 0.2083
Epoch 13/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.1617 - accuracy: 0.2069 - val_loss: 0.1611 - val_accuracy: 0.2116
Epoch 14/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.1611 - accuracy: 0.2093 - val_loss: 0.1604 - val_accuracy: 0.2128
Epoch 15/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.1604 - accuracy: 0.2120 - val_loss: 0.1599 - val_accuracy: 0.2138
Epoch 16/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.1599 - accuracy: 0.2137 - val_loss: 0.1595 - val_accuracy: 0.2138
Epoch 17/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.1594 - accuracy: 0.2150 - val_loss: 0.1591 - val_accuracy: 0.2145
Epoch 18/20
1250/1250 [==============================] - 6s 4ms/step - loss: 0.1590 - accuracy: 0.2156 - val_loss: 0.1587 - val_accuracy: 0.2167
Epoch 19/20
1250/1250 [==============================] - 6s 5ms/step - loss: 0.1586 - accuracy: 0.2176 - val_loss: 0.1583 - val_accuracy: 0.2183
Epoch 20/20
1250/1250 [==============================] - 6s 4ms/step - loss: 0.1582 - accuracy: 0.2185 - val_loss: 0.1581 - val_accuracy: 0.2212
```

```
Training time: 117.8701229095459s
test loss:  0.1581224799156189
test acc:  21.799999475479126
f1 score:   0.13175661301228325
recall score:  0.218
precision score:  0.10805050960068179
confusion matrix:

 [[640  99   0   0   0 103  16   0   0 142]
 [133 356   0   0   0  97   7   0   0 407]
 [569  93   0   0   0 224  21   0   0  93]
 [269 100   0   0   0 419  35   0   0 177]
 [460  94   0   0   0 343  31   0   0  72]
 [299  62   0   0   0 507  21   0   0 111]
 [314 179   0   0   0 312  37   0   0 158]
 [385 112   0   0   0 248  42   0   0 213]
 [419 126   0   0   0 171   6   0   0 278]
 [142 147   0   0   0  56  15   0   0 640]]
```





As you can see the best model is belongs to categorical_crossentropy Loss function.

2.1.7 OPTIMIZER ANALYSE:

In this part we have chosen softmax for the last layer and then try different optimizer for models.

Optimizer : SGD(momentum)

Architecture of neural net :

In this part I have used **four layers** neural net which consist **two hidden layers**. At the final layer I have used **softmax** as of our activation-function, **Relu** as of **second** and **third** activation layer.

I have used **512 neurons** in first layer , **512 neurons** in second layer and **512 neurons** in third layer and **10 neurons** at the final layer.

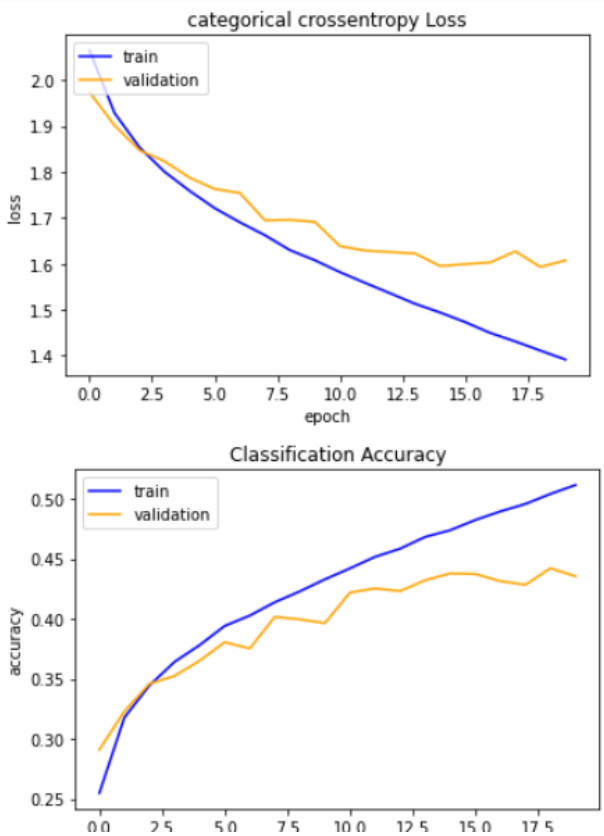
Results:

Training process :

```
Epoch 1/20
1250/1250 [==============================] - 7s 5ms/step - loss: 2.0644 - accuracy: 0.2547 - val_loss: 1.9736 - val_accuracy: 0.2909
Epoch 2/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.9282 - accuracy: 0.3177 - val_loss: 1.9012 - val_accuracy: 0.3229
Epoch 3/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.8531 - accuracy: 0.3444 - val_loss: 1.8470 - val_accuracy: 0.3458
Epoch 4/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.8001 - accuracy: 0.3641 - val_loss: 1.8230 - val_accuracy: 0.3523
Epoch 5/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.7586 - accuracy: 0.3780 - val_loss: 1.7873 - val_accuracy: 0.3650
Epoch 6/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.7210 - accuracy: 0.3940 - val_loss: 1.7628 - val_accuracy: 0.3804
Epoch 7/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.6904 - accuracy: 0.4026 - val_loss: 1.7543 - val_accuracy: 0.3752
Epoch 8/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6622 - accuracy: 0.4137 - val_loss: 1.6947 - val_accuracy: 0.4015
Epoch 9/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6297 - accuracy: 0.4228 - val_loss: 1.6956 - val_accuracy: 0.3994
Epoch 10/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6076 - accuracy: 0.4328 - val_loss: 1.6912 - val_accuracy: 0.3963
Epoch 11/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5817 - accuracy: 0.4419 - val_loss: 1.6386 - val_accuracy: 0.4218
Epoch 12/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5585 - accuracy: 0.4516 - val_loss: 1.6287 - val_accuracy: 0.4252
Epoch 13/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5358 - accuracy: 0.4583 - val_loss: 1.6256 - val_accuracy: 0.4230
Epoch 13/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5358 - accuracy: 0.4583 - val_loss: 1.6256 - val_accuracy: 0.4230
Epoch 14/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5129 - accuracy: 0.4681 - val_loss: 1.6225 - val_accuracy: 0.4320
Epoch 15/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.4939 - accuracy: 0.4737 - val_loss: 1.5950 - val_accuracy: 0.4377
Epoch 16/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4730 - accuracy: 0.4822 - val_loss: 1.5993 - val_accuracy: 0.4372
Epoch 17/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.4494 - accuracy: 0.4894 - val_loss: 1.6029 - val_accuracy: 0.4313
Epoch 18/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4310 - accuracy: 0.4956 - val_loss: 1.6271 - val_accuracy: 0.4283
Epoch 19/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.4111 - accuracy: 0.5040 - val_loss: 1.5931 - val_accuracy: 0.4420
Epoch 20/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.3915 - accuracy: 0.5114 - val_loss: 1.6073 - val_accuracy: 0.4353
313/313 [==============================] - 1s 4ms/step - loss: 1.6066 - accuracy: 0.4286
```

```
Training time: 127.27400970458984s
test loss:  1.6065670251846313
test acc:  42.8600013256073
f1 score:   0.4178009251319998
recall score:  0.4286
precision score:  0.44461876752009205
confusion matrix:

[[229  88 198  15  65  12  46  56 244  47]
[  3 663  15  15  18  10  36  17  92 131]
[ 15  48 482  58  93  64  95  63  56  26]
[ 11  61 144 225  56 143 155  79  53  73]
[ 21  52 271  48 276  29 116  95  68  24]
[ 11  29 163 152  58 323  90  82  49  43]
[  7  92 132  59  75  32 481  41  42  39]
[ 10  53 102  51  59  69  42 495  53  66]
[ 35 125  41  18  27  18  20  29 631  56]
[  7 282  28  20  13  16  25  54  74 481]]
```



Optimizer : Adam

Architecture of neural net :

In this part I have used **four layers** neural net which consist **two hidden layers**. At the final layer I have used **softmax** as of our activation-function, **Relu** as of **second** and **third** activation layer.

I have used **512 neurons** in first layer , **512 neurons** in second layer and **512 neurons** in third layer and **10 neurons** at the final layer.

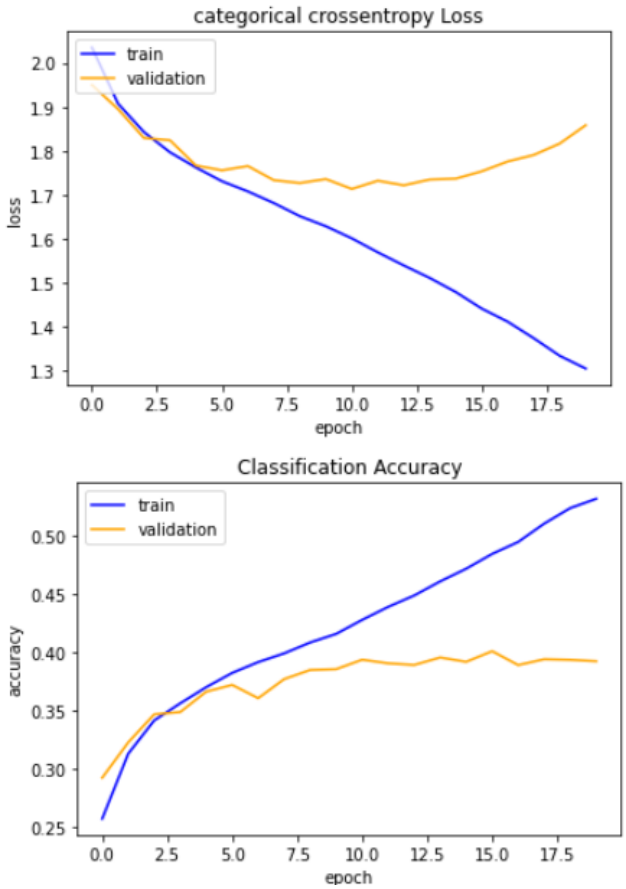
Results:

Training process :

```
...  Epoch 1/20
     1250/1250 [==============================] - 8s 5ms/step - loss: 2.0345 - accuracy: 0.2571 - val_loss: 1.9496 - val_accuracy: 0.2921
     Epoch 2/20
     1250/1250 [==============================] - 7s 5ms/step - loss: 1.9074 - accuracy: 0.3127 - val_loss: 1.8949 - val_accuracy: 0.3224
     Epoch 3/20
     1250/1250 [==============================] - 6s 5ms/step - loss: 1.8426 - accuracy: 0.3413 - val_loss: 1.8291 - val_accuracy: 0.3465
     Epoch 4/20
     1250/1250 [==============================] - 7s 5ms/step - loss: 1.7966 - accuracy: 0.3559 - val_loss: 1.8240 - val_accuracy: 0.3484
     Epoch 5/20
     1250/1250 [==============================] - 6s 5ms/step - loss: 1.7620 - accuracy: 0.3698 - val_loss: 1.7665 - val_accuracy: 0.3659
     Epoch 6/20
     1250/1250 [==============================] - 7s 5ms/step - loss: 1.7308 - accuracy: 0.3820 - val_loss: 1.7556 - val_accuracy: 0.3717
     Epoch 7/20
     1250/1250 [==============================] - 7s 5ms/step - loss: 1.7076 - accuracy: 0.3913 - val_loss: 1.7653 - val_accuracy: 0.3603
     Epoch 8/20
     1250/1250 [==============================] - 6s 5ms/step - loss: 1.6811 - accuracy: 0.3988 - val_loss: 1.7331 - val_accuracy: 0.3767
     Epoch 9/20
     1250/1250 [==============================] - 6s 5ms/step - loss: 1.6513 - accuracy: 0.4081 - val_loss: 1.7267 - val_accuracy: 0.3845
     Epoch 10/20
     1250/1250 [==============================] - 7s 5ms/step - loss: 1.6285 - accuracy: 0.4155 - val_loss: 1.7357 - val_accuracy: 0.3852
     Epoch 11/20
     1250/1250 [==============================] - 7s 5ms/step - loss: 1.6009 - accuracy: 0.4275 - val_loss: 1.7133 - val_accuracy: 0.3933
     Epoch 12/20
     1250/1250 [==============================] - 7s 5ms/step - loss: 1.5691 - accuracy: 0.4386 - val_loss: 1.7319 - val_accuracy: 0.3902
     Epoch 13/20
     1250/1250 [==============================] - 7s 5ms/step - loss: 1.5393 - accuracy: 0.4484 - val_loss: 1.7215 - val_accuracy: 0.3888
     Epoch 14/20
     1250/1250 [==============================] - 7s 5ms/step - loss: 1.5108 - accuracy: 0.4605 - val_loss: 1.7346 - val_accuracy: 0.3952
     Epoch 15/20
     1250/1250 [==============================] - 6s 5ms/step - loss: 1.4789 - accuracy: 0.4713 - val_loss: 1.7367 - val_accuracy: 0.3915
     Epoch 16/20
     1250/1250 [==============================] - 7s 5ms/step - loss: 1.4412 - accuracy: 0.4840 - val_loss: 1.7533 - val_accuracy: 0.4006
     Epoch 17/20
     1250/1250 [==============================] - 6s 5ms/step - loss: 1.4114 - accuracy: 0.4943 - val_loss: 1.7757 - val_accuracy: 0.3887
     Epoch 18/20
     1250/1250 [==============================] - 6s 5ms/step - loss: 1.3739 - accuracy: 0.5100 - val_loss: 1.7905 - val_accuracy: 0.3937
     Epoch 19/20
     1250/1250 [==============================] - 7s 5ms/step - loss: 1.3339 - accuracy: 0.5233 - val_loss: 1.8164 - val_accuracy: 0.3932
     Epoch 20/20
     1250/1250 [==============================] - 6s 5ms/step - loss: 1.3049 - accuracy: 0.5312 - val_loss: 1.8581 - val_accuracy: 0.3920
     313/313 [==============================] - 1s 4ms/step - loss: 1.8554 - accuracy: 0.3982
```

```
Training time: 132.68761730194092s
test loss:  1.8554269075393677
test acc:  39.820000529289246
f1 score:   0.3972877461903911
recall score:  0.3982
precision score:  0.4022676522753314
confusion matrix:

 [[384  52 121  26 103  23  82  52 123  34]
 [ 44 480  24  44  42  32  56  25 104 149]
 [ 69  25 344  97 156  73 126  56  40  14]
 [ 39  36  94 259 104 137 176  63  39  53]
 [ 57  29 159  56 368  35 147  93  44  12]
 [ 30  20  85 156 123 298 132  77  50  29]
 [ 33  39  83  64 132  55 504  38  23  29]
 [ 59  42  63  56 113  72  77 434  38  46]
 [137 101  27  34  51  28  38  39 502  43]
 [ 46 196  23  48  30  36  61  39 112 409]]
```



Optimizer : Adamax

Architecture of neural net :

In this part I have used **four layers** neural net which consist **two hidden layers**. At the final layer I have used **softmax** as of our activation-function, **Relu** as of **second** and **third** activation layer.

I have used **512 neurons** in first layer , **512 neurons** in second layer and **512 neurons** in third layer and **10 neurons** at the final layer.

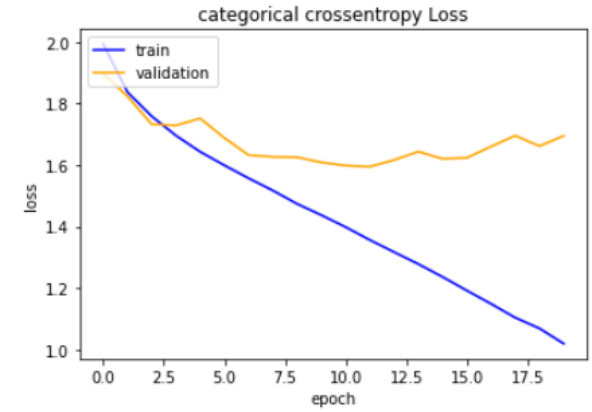
Results:

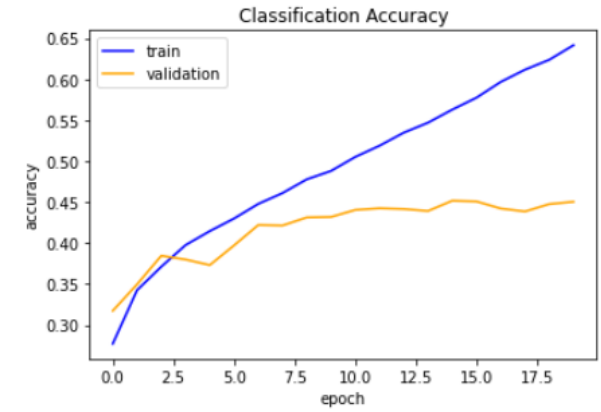
Training process :

```
Epoch 1/20
1250/1250 [==============================] - 8s 6ms/step - loss: 1.9933 - accuracy: 0.2774 - val_loss: 1.8966 - val_accuracy: 0.3175
Epoch 2/20
1250/1250 [==============================] - 7s 6ms/step - loss: 1.8373 - accuracy: 0.3426 - val_loss: 1.8236 - val_accuracy: 0.3497
Epoch 3/20
1250/1250 [==============================] - 7s 6ms/step - loss: 1.7587 - accuracy: 0.3713 - val_loss: 1.7327 - val_accuracy: 0.3846
Epoch 4/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.6959 - accuracy: 0.3977 - val_loss: 1.7292 - val_accuracy: 0.3800
Epoch 5/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.6431 - accuracy: 0.4147 - val_loss: 1.7518 - val_accuracy: 0.3732
Epoch 6/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5990 - accuracy: 0.4300 - val_loss: 1.6881 - val_accuracy: 0.3972
Epoch 7/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.5569 - accuracy: 0.4479 - val_loss: 1.6323 - val_accuracy: 0.4221
Epoch 8/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.5170 - accuracy: 0.4611 - val_loss: 1.6268 - val_accuracy: 0.4215
Epoch 9/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.4736 - accuracy: 0.4778 - val_loss: 1.6254 - val_accuracy: 0.4313
Epoch 10/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.4368 - accuracy: 0.4881 - val_loss: 1.6087 - val_accuracy: 0.4319
Epoch 11/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.3980 - accuracy: 0.5052 - val_loss: 1.5983 - val_accuracy: 0.4405
Epoch 12/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.3560 - accuracy: 0.5188 - val_loss: 1.5946 - val_accuracy: 0.4425
Epoch 13/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.3166 - accuracy: 0.5347 - val_loss: 1.6163 - val_accuracy: 0.4416
Epoch 14/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.2777 - accuracy: 0.5469 - val_loss: 1.6437 - val_accuracy: 0.4392
Epoch 15/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.2358 - accuracy: 0.5628 - val_loss: 1.6205 - val_accuracy: 0.4517
Epoch 16/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.1915 - accuracy: 0.5774 - val_loss: 1.6234 - val_accuracy: 0.4508
Epoch 17/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.1484 - accuracy: 0.5966 - val_loss: 1.6602 - val_accuracy: 0.4422
Epoch 18/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.1032 - accuracy: 0.6116 - val_loss: 1.6957 - val_accuracy: 0.4386
Epoch 19/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.0682 - accuracy: 0.6236 - val_loss: 1.6617 - val_accuracy: 0.4476
Epoch 20/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.0185 - accuracy: 0.6414 - val_loss: 1.6950 - val_accuracy: 0.4504
313/313 [==============================] - 1s 4ms/step - loss: 1.6942 - accuracy: 0.4448
```

```
Training time: 134.8786997795105s
test loss:  1.6941689252853394
test acc:  44.47999894618988
f1 score:   0.44988474686413804
recall score:  0.4448
precision score:  0.46306475159452304
confusion matrix:

 [[435  33 119  34 137  16  53  39  95  39]
 [ 36 525  30  56  39  20  55  21  72 146]
 [ 55  18 404 115 189  43  87  45  30  14]
 [ 36  20 117 352 119 125 124  47  23  37]
 [ 40   8 156 107 420  42 114  75  23  15]
 [ 25   8 131 219  93 329  82  64  24  25]
 [ 19  29  99 117 112  43 506  28  19  28]
 [ 30  12  80  91 126  59  33 510  21  38]
 [117  77  49  55  75  15  38  29 485  60]
 [ 42 150  35  78  34  23  51  37  68 482]]
```





As you can see the best optimizer is belongs to Adamax.

2.1.8 NUMBER OF LAYERS ANALYSE:

In this part we have chosen softmax for the last layer and then try different number of layers for models.

Number of layer : 3

Architecture of neural net :

In this part I have used **three layers** neural net which consist **one hidden layers**. At the final layer I have used **softmax** as of our activation-function, **Relu** as of **second** activation layer.

I have used **512 neurons** in first layer , **512 neurons** in second layer and and **10 neurons** at the final layer.

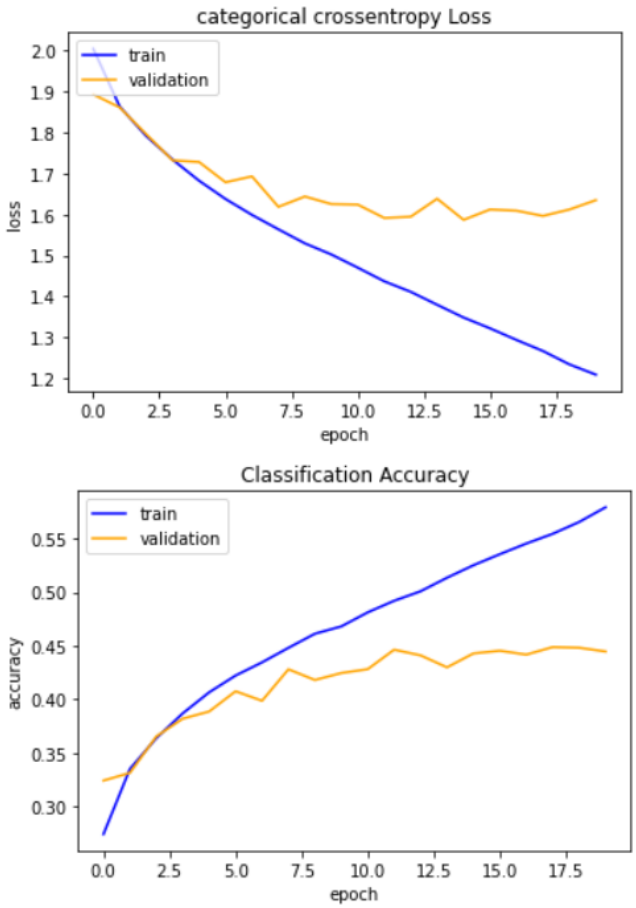
Results:

Training process :

```
... Epoch 1/20
    1250/1250 [==============================] - 9s 5ms/step - loss: 2.0032 - accuracy: 0.2740 - val_loss: 1.8920 - val_accuracy: 0.3240
    Epoch 2/20
    1250/1250 [==============================] - 6s 5ms/step - loss: 1.8624 - accuracy: 0.3349 - val_loss: 1.8604 - val_accuracy: 0.3312
    Epoch 3/20
    1250/1250 [==============================] - 6s 5ms/step - loss: 1.7908 - accuracy: 0.3637 - val_loss: 1.7953 - val_accuracy: 0.3652
    Epoch 4/20
    1250/1250 [==============================] - 6s 5ms/step - loss: 1.7333 - accuracy: 0.3868 - val_loss: 1.7315 - val_accuracy: 0.3818
    Epoch 5/20
    1250/1250 [==============================] - 6s 5ms/step - loss: 1.6819 - accuracy: 0.4065 - val_loss: 1.7275 - val_accuracy: 0.3885
    Epoch 6/20
    1250/1250 [==============================] - 6s 5ms/step - loss: 1.6374 - accuracy: 0.4221 - val_loss: 1.6781 - val_accuracy: 0.4074
    Epoch 7/20
    1250/1250 [==============================] - 6s 5ms/step - loss: 1.5987 - accuracy: 0.4344 - val_loss: 1.6925 - val_accuracy: 0.3985
    Epoch 8/20
    1250/1250 [==============================] - 6s 5ms/step - loss: 1.5635 - accuracy: 0.4480 - val_loss: 1.6183 - val_accuracy: 0.4279
    Epoch 9/20
    1250/1250 [==============================] - 6s 5ms/step - loss: 1.5293 - accuracy: 0.4611 - val_loss: 1.6435 - val_accuracy: 0.4179
    Epoch 10/20
    1250/1250 [==============================] - 6s 5ms/step - loss: 1.5023 - accuracy: 0.4680 - val_loss: 1.6250 - val_accuracy: 0.4244
    Epoch 11/20
    1250/1250 [==============================] - 6s 5ms/step - loss: 1.4701 - accuracy: 0.4811 - val_loss: 1.6237 - val_accuracy: 0.4280
    Epoch 12/20
    1250/1250 [==============================] - 6s 5ms/step - loss: 1.4370 - accuracy: 0.4919 - val_loss: 1.5912 - val_accuracy: 0.4462
    Epoch 13/20
    1250/1250 [==============================] - 6s 5ms/step - loss: 1.4113 - accuracy: 0.5008 - val_loss: 1.5942 - val_accuracy: 0.4409
    Epoch 14/20
    1250/1250 [==============================] - 6s 5ms/step - loss: 1.3798 - accuracy: 0.5134 - val_loss: 1.6381 - val_accuracy: 0.4297
    Epoch 15/20
    1250/1250 [==============================] - 6s 5ms/step - loss: 1.3487 - accuracy: 0.5251 - val_loss: 1.5866 - val_accuracy: 0.4427
Epoch 15/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.3487 - accuracy: 0.5251 - val_loss: 1.5866 - val_accuracy: 0.4427
Epoch 16/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.3224 - accuracy: 0.5354 - val_loss: 1.6118 - val_accuracy: 0.4452
Epoch 17/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.2941 - accuracy: 0.5453 - val_loss: 1.6089 - val_accuracy: 0.4416
Epoch 18/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.2672 - accuracy: 0.5544 - val_loss: 1.5961 - val_accuracy: 0.4486
Epoch 19/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.2346 - accuracy: 0.5655 - val_loss: 1.6118 - val_accuracy: 0.4480
Epoch 20/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.2095 - accuracy: 0.5792 - val_loss: 1.6346 - val_accuracy: 0.4445
313/313 [==============================] - 1s 3ms/step - loss: 1.6388 - accuracy: 0.4399
```

```
Training time: 142.69283962249756s
test loss:  1.6387954950332642
test acc:  43.99000108242035
f1 score:   0.43295811309082793
recall score:  0.4399
precision score:  0.44278888074717376
confusion matrix:

 [[336  70 131  35  44  30  48  77 177  52]
 [ 21 632  10  19   7  16  32  28  74 161]
 [ 35  32 408 112  84  74  82  93  52  28]
 [ 28  61  85 302  48 163 107 101  37  68]
 [ 38  43 179 101 230  55 133 138  53  30]
 [ 16  33  94 216  29 365  66 108  32  41]
 [ 14  83  93 102  39  69 484  60  21  35]
 [ 16  27  53  69  34  59  38 600  36  68]
 [ 77 122  23  40  22  23  16  53 554  70]
 [ 19 259  25  42   7  14  22  65  59 488]]
```



Number of layer : 4

Architecture of neural net :

In this part I have used **four layers** neural net which consist **two hidden layers**. At the final layer I have used **softmax** as of our activation-function, **Relu** as of **second and third** activation layer.

I have used **512 neurons** in first layer , **512 neurons** in second and third layer and and **10 neurons** at the final layer.

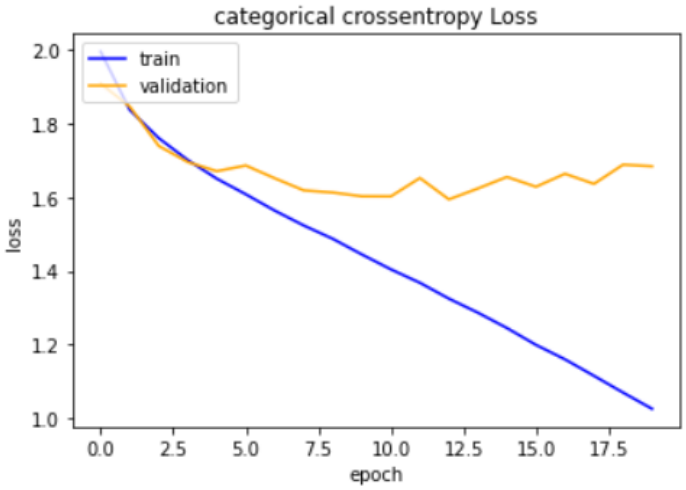
Results:

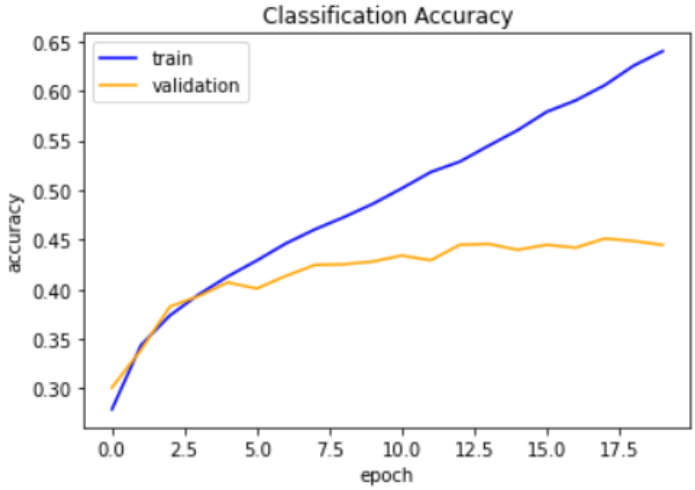
Training process :

```
Epoch 1/20
1250/1250 [==============================] - 9s 5ms/step - loss: 1.9963 - accuracy: 0.2791 - val_loss: 1.9094 - val_accuracy: 0.3009
Epoch 2/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.8381 - accuracy: 0.3437 - val_loss: 1.8470 - val_accuracy: 0.3389
Epoch 3/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.7612 - accuracy: 0.3737 - val_loss: 1.7399 - val_accuracy: 0.3825
Epoch 4/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.7016 - accuracy: 0.3950 - val_loss: 1.6961 - val_accuracy: 0.3933
Epoch 5/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.6509 - accuracy: 0.4131 - val_loss: 1.6713 - val_accuracy: 0.4070
Epoch 6/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6084 - accuracy: 0.4291 - val_loss: 1.6868 - val_accuracy: 0.4008
Epoch 7/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.5639 - accuracy: 0.4463 - val_loss: 1.6519 - val_accuracy: 0.4135
Epoch 8/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5235 - accuracy: 0.4605 - val_loss: 1.6190 - val_accuracy: 0.4246
Epoch 9/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4872 - accuracy: 0.4728 - val_loss: 1.6131 - val_accuracy: 0.4253
Epoch 10/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.4448 - accuracy: 0.4862 - val_loss: 1.6030 - val_accuracy: 0.4280
Epoch 11/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4042 - accuracy: 0.5018 - val_loss: 1.6028 - val_accuracy: 0.4341
Epoch 12/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.3678 - accuracy: 0.5183 - val_loss: 1.6527 - val_accuracy: 0.4293
Epoch 13/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.3246 - accuracy: 0.5288 - val_loss: 1.5946 - val_accuracy: 0.4449
Epoch 14/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.2864 - accuracy: 0.5449 - val_loss: 1.6242 - val_accuracy: 0.4458
Epoch 15/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.2446 - accuracy: 0.5606 - val_loss: 1.6556 - val_accuracy: 0.4399
Epoch 16/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.1988 - accuracy: 0.5791 - val_loss: 1.6289 - val_accuracy: 0.4449
Epoch 17/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.1595 - accuracy: 0.5906 - val_loss: 1.6642 - val_accuracy: 0.4420
Epoch 18/20
1250/1250 [==============================] - 7s 5ms/step - loss: 1.1147 - accuracy: 0.6058 - val_loss: 1.6367 - val_accuracy: 0.4512
Epoch 19/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.0692 - accuracy: 0.6256 - val_loss: 1.6890 - val_accuracy: 0.4487
Epoch 20/20
1250/1250 [==============================] - 6s 5ms/step - loss: 1.0249 - accuracy: 0.6403 - val_loss: 1.6847 - val_accuracy: 0.4448
313/313 [==============================] - 1s 3ms/step - loss: 1.6925 - accuracy: 0.4402
```

```
Training time: 142.83227610588074s
test loss:  1.692490577697754
test acc:  44.0200001001358
f1 score:   0.4395874329203775
recall score:  0.4402
precision score:  0.44547390493581307
confusion matrix:

[[404  40 169  31  78  32  35  46 125  40]
 [ 31 587  15  30  22  21  45  49  75 125]
 [ 55  17 454  94 100  92  73  74  33   8]
 [ 29  34 133 257  76 194 131  77  30  39]
 [ 44  17 245  64 298  63 101 115  37  16]
 [ 22  17 136 152  55 391  75 112  24  16]
 [ 28  46 147  62  84  61 475  49  21  27]
 [ 33  18  79  73  70  79  33 565  24  26]
 [101  93  43  31  51  47  22  38 505  69]
 [ 38 189  35  57  21  31  44  52  67 466]]
```





Number of layer : 5

Architecture of neural net :

In this part I have used **five layers** neural net which consist **three hidden layers**. At the final layer I have used **softmax** as of our activation-function, **Relu** as of **second and third and fourth** activation layer.

I have used **512 neurons** in first layer , **512 neurons** in second layer and **512 neurons** in third layer and **512 neurons** and **10 neurons** at the final layer.

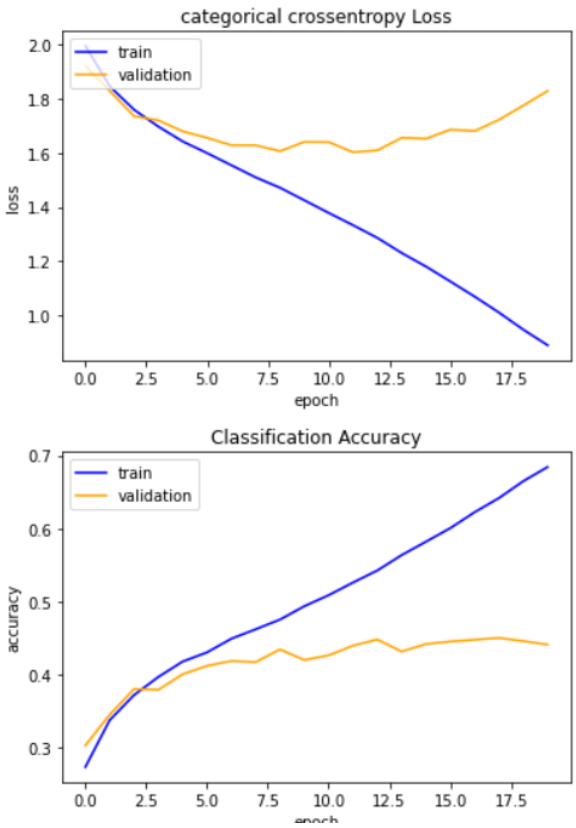
Results:

Training process :

```
Epoch 1/20
1250/1250 [==============================] - 8s 6ms/step - loss: 1.9930 - accuracy: 0.2736 - val_loss: 1.9200 - val_accuracy: 0.3030
Epoch 2/20
1250/1250 [==============================] - 7s 6ms/step - loss: 1.8447 - accuracy: 0.3376 - val_loss: 1.8274 - val_accuracy: 0.3451
Epoch 3/20
1250/1250 [==============================] - 8s 6ms/step - loss: 1.7584 - accuracy: 0.3720 - val_loss: 1.7348 - val_accuracy: 0.3802
Epoch 4/20
1250/1250 [==============================] - 7s 6ms/step - loss: 1.6963 - accuracy: 0.3968 - val_loss: 1.7195 - val_accuracy: 0.3792
Epoch 5/20
1250/1250 [==============================] - 7s 6ms/step - loss: 1.6416 - accuracy: 0.4179 - val_loss: 1.6789 - val_accuracy: 0.4005
Epoch 6/20
1250/1250 [==============================] - 8s 6ms/step - loss: 1.5988 - accuracy: 0.4304 - val_loss: 1.6552 - val_accuracy: 0.4119
Epoch 7/20
1250/1250 [==============================] - 8s 6ms/step - loss: 1.5539 - accuracy: 0.4491 - val_loss: 1.6279 - val_accuracy: 0.4186
Epoch 8/20
1250/1250 [==============================] - 7s 6ms/step - loss: 1.5092 - accuracy: 0.4620 - val_loss: 1.6277 - val_accuracy: 0.4170
Epoch 9/20
1250/1250 [==============================] - 7s 6ms/step - loss: 1.4710 - accuracy: 0.4752 - val_loss: 1.6060 - val_accuracy: 0.4345
Epoch 10/20
1250/1250 [==============================] - 7s 6ms/step - loss: 1.4247 - accuracy: 0.4935 - val_loss: 1.6405 - val_accuracy: 0.4199
Epoch 11/20
1250/1250 [==============================] - 8s 6ms/step - loss: 1.3783 - accuracy: 0.5086 - val_loss: 1.6395 - val_accuracy: 0.4263
Epoch 12/20
1250/1250 [==============================] - 8s 6ms/step - loss: 1.3332 - accuracy: 0.5258 - val_loss: 1.6020 - val_accuracy: 0.4395
Epoch 13/20
1250/1250 [==============================] - 8s 6ms/step - loss: 1.2864 - accuracy: 0.5424 - val_loss: 1.6091 - val_accuracy: 0.4480
Epoch 14/20
1250/1250 [==============================] - 7s 6ms/step - loss: 1.2307 - accuracy: 0.5634 - val_loss: 1.6554 - val_accuracy: 0.4315
Epoch 15/20
1250/1250 [==============================] - 7s 6ms/step - loss: 1.1806 - accuracy: 0.5818 - val_loss: 1.6521 - val_accuracy: 0.4417
Epoch 16/20
1250/1250 [==============================] - 8s 6ms/step - loss: 1.1256 - accuracy: 0.6004 - val_loss: 1.6855 - val_accuracy: 0.4452
Epoch 17/20
1250/1250 [==============================] - 8s 6ms/step - loss: 1.0699 - accuracy: 0.6221 - val_loss: 1.6805 - val_accuracy: 0.4477
Epoch 18/20
1250/1250 [==============================] - 7s 6ms/step - loss: 1.0111 - accuracy: 0.6415 - val_loss: 1.7219 - val_accuracy: 0.4501
Epoch 19/20
1250/1250 [==============================] - 7s 6ms/step - loss: 0.9482 - accuracy: 0.6646 - val_loss: 1.7745 - val_accuracy: 0.4458
Epoch 20/20
1250/1250 [==============================] - 7s 6ms/step - loss: 0.8906 - accuracy: 0.6840 - val_loss: 1.8282 - val_accuracy: 0.4410
313/313 [==============================] - 1s 4ms/step - loss: 1.8323 - accuracy: 0.4372
```

```
Training time: 150.58553385734558s
test loss:  1.832317590713501
test acc:  43.720000982284546
f1 score:   0.4363100682804098
recall score:  0.4372
precision score:  0.44436853601895165
confusion matrix:

 [[542  27  90  22  74  19  36  27 142  21]
 [ 75 514  16  38  33  23  54  14 103 130]
 [109  20 376  91 152  57  78  46  56  15]
 [ 65  30  89 249 120 150 147  61  50  39]
 [114   9 178  51 375  39 108  55  65   6]
 [ 67  14 107 178  81 336  88  64  52  13]
 [ 69  28  93  64 121  36 496  26  43  24]
 [ 90  10  63  70 103  64  41 483  44  32]
 [142  65  12  25  55  36  27  20 583  35]
 [ 80 157  15  55  32  31  41  60 111 418]]
```



As you can see the best model is belongs to model with **four layers.**

2.1.9 BEST PARAMETERS , BEST PERFORMANCE :

In this part we want to introduce the best parameters which leads to best performance according to our process in the previous section :

As we found, the number of neurons is a critical parameter to let us achieve best model and so we did a exact and adequate process to calculate the best value.

In this case we set the number of neurons to **512 in hidden layer** which give us pretty much better performance.

One more thing I want to introduce about my implementation is about batch-size which has been set to **32** which leads to a pretty good performance.

And also as we found the best number of layer is equal to **four**.

2.1.10 REPORTING THE BEST MODEL PARAMETERS:

Number of Neuron	Batch-Size	Activation-function	Loss	Optimizer	Number-Layer
512	32	ReLU	Cross-entropy	Adamax	4

If there is any issue with that my solution, please let me know to provide more detailed information. (Be in touch **here!**)

For running this part, please have an accurate look on related directory and There you can easily find all of you need and also, I would appreciate it if you would consider them ☺

3 QUESTION #2

3.1 MULTI-LAYER PERCEPTRON (REGRESSION)

In this part we intend to predict the price of house based on a regression problem.

I think the most important part of these problem is about pre-processing so first of all we express the ideas about pre-processing.

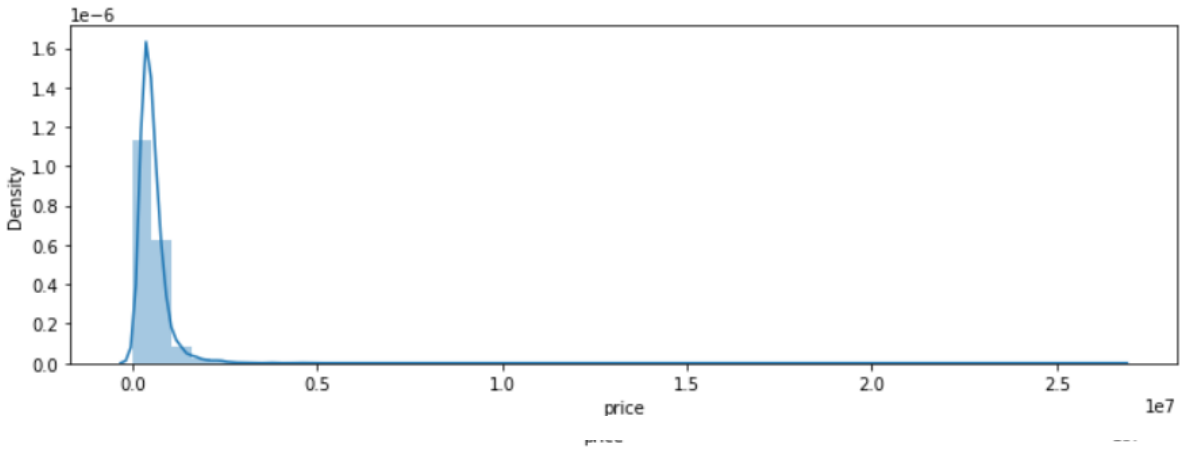
3.1.1 THE PRE-PROCESSING STAGE ON DATASET

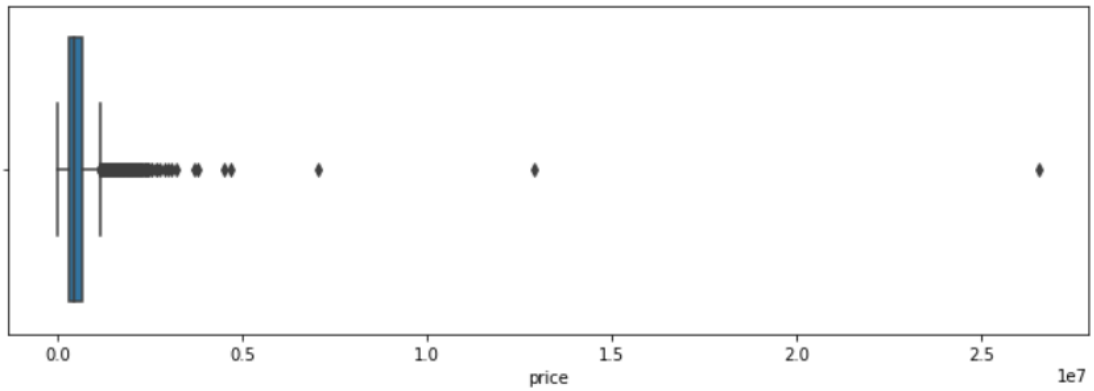
First of all we investigate dataset using .info command in python to find out how many feature we have and what are the types of data rows and then the result is similar to :

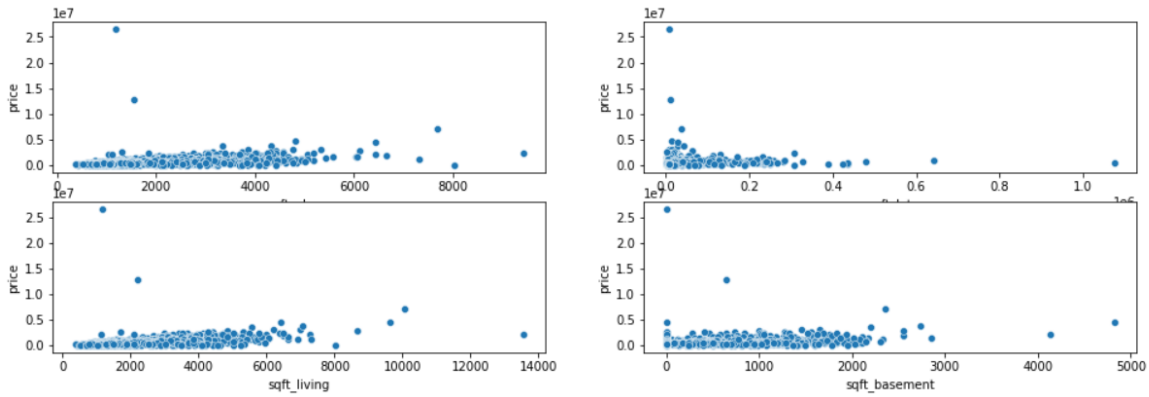
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4600 entries, 0 to 4599
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   date           4600 non-null   object
 1   price          4600 non-null   float64
 2   bedrooms       4600 non-null   float64
 3   bathrooms      4600 non-null   float64
 4   sqft_living    4600 non-null   int64
 5   sqft_lot       4600 non-null   int64
 6   floors         4600 non-null   float64
 7   waterfront     4600 non-null   int64
 8   view           4600 non-null   int64
 9   condition      4600 non-null   int64
 10  sqft_above     4600 non-null   int64
 11  sqft_basement  4600 non-null   int64
 12  yr_built       4600 non-null   int64
 13  yr_renovated   4600 non-null   int64
 14  street         4600 non-null   object
 15  city           4600 non-null   object
 16  statezip       4600 non-null   object
 17  country        4600 non-null   object
dtypes: float64(4), int64(9), object(5)
memory usage: 647.0+ KB
```

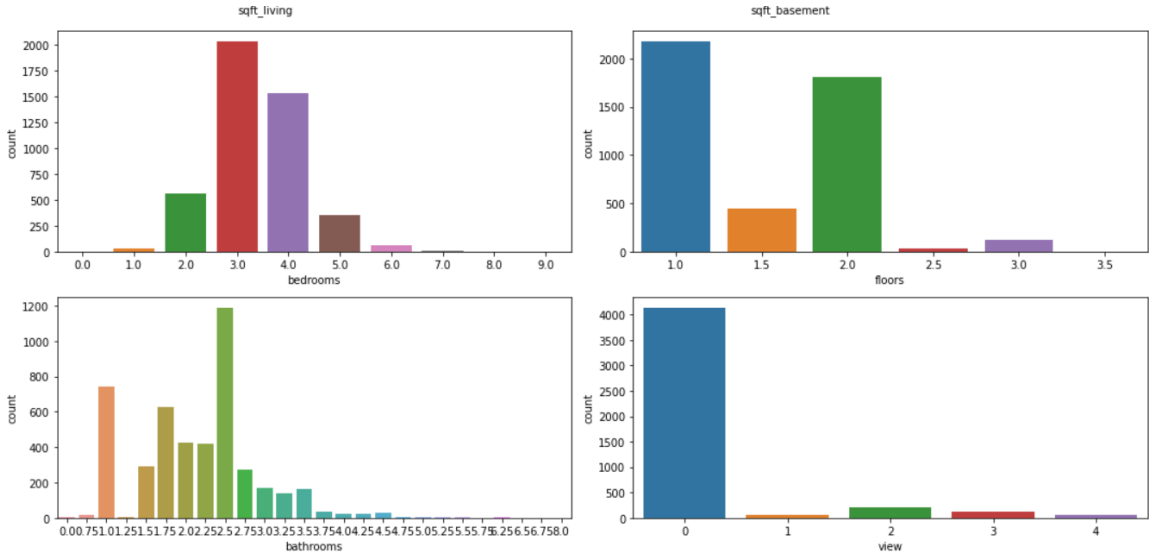
	count	mean	std	min	25%	50%	75%	max
price	4600.0	551962.988473	563834.702547	0.0	322875.00	460943.461539	654962.50	26590000.0
bedrooms	4600.0	3.400870	0.908848	0.0	3.00	3.000000	4.00	9.0
bathrooms	4600.0	2.160815	0.783781	0.0	1.75	2.250000	2.50	8.0
sqft_living	4600.0	2139.346957	963.206916	370.0	1460.00	1980.000000	2620.00	13540.0
sqft_lot	4600.0	14852.516087	35884.436145	638.0	5000.75	7683.000000	11001.25	1074218.0
floors	4600.0	1.512065	0.538288	1.0	1.00	1.500000	2.00	3.5
waterfront	4600.0	0.007174	0.084404	0.0	0.00	0.000000	0.00	1.0
view	4600.0	0.240652	0.778405	0.0	0.00	0.000000	0.00	4.0
condition	4600.0	3.451739	0.677230	1.0	3.00	3.000000	4.00	5.0
sqft_above	4600.0	1827.265435	862.168977	370.0	1190.00	1590.000000	2300.00	9410.0
sqft_basement	4600.0	312.081522	464.137228	0.0	0.00	0.000000	610.00	4820.0
yr_built	4600.0	1970.786304	29.731848	1900.0	1951.00	1976.000000	1997.00	2014.0
yr_renovated	4600.0	808.608261	979.414536	0.0	0.00	0.000000	1999.00	2014.0

After that we have done some calculations to get a better insight :










```
# check if there are any Null values
Data.isnull().sum()
```

```
date           0
price          0
bedrooms       0
bathrooms      0
sqft_living    0
sqft_lot       0
floors         0
waterfront     0
view           0
condition      0
sqft_above     0
sqft_basement  0
yr_built       0
yr_renovated   0
street         0
city           0
statezip       0
country        0
dtype: int64
```

I have used mean instead of zero values in price column to get a better performance as of a another pre-processing stage.

And also I have used Labelencoder() to convert objects to the numerical data.

3.1.2 SPLITTING DATA

In this part I have allocated 80 percent of data to the train and rest of that for the test.

3.1.3 DESIGNING BEST MODEL

In this part we have tested four model to obtain the best fit and further you can see that I have reported the performance for all modes.

But as you can see we prefer the ReLU and also three layer according to the rest results.

Model Selection and Evaluation

Model One : 2 layers with ReLu as of activation function

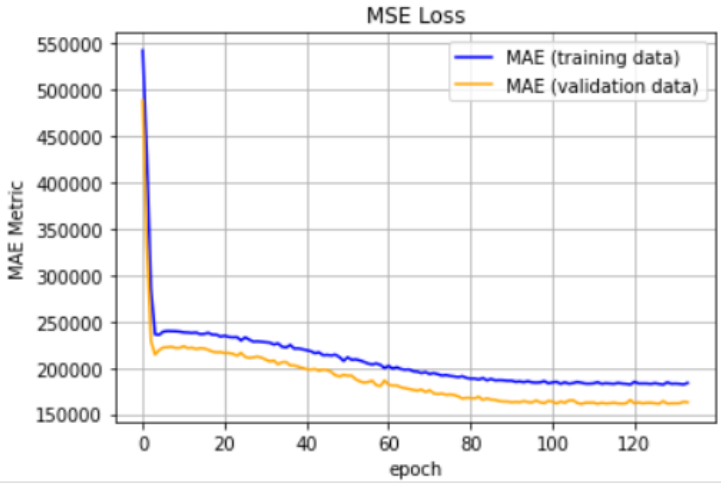
```
[ ] model = Sequential()
    model.add(Dense(10, activation='relu',input_shape=(15,),kernel_initializer='normal'))
    model.add(Dense(1, kernel_initializer='normal'))
    model.compile(loss='mean_squared_error', optimizer='adam',  metrics=['mse', 'mae'])
    from keras import callbacks
    earlystopping = callbacks.EarlyStopping(monitor ="val_loss", mode ="min", patience = 5, restore_best_weights = True,verbose=1)
    trained_model = model.fit(X_train,y_train, batch_size=16, epochs=150, validation_split=0.15 ,callbacks =[earlystopping])
    history = trained_model.history
```

evalute number of neurons and layers

```
# history
Y_pred = model.predict(X_test)
y_pred=np.ravel(Y_pred)
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('VarScore:',metrics.explained_variance_score(y_test,y_pred))
```

```
MAE: 178404.83
MSE: 255575100000.0
VarScore: 0.177839457988739
```

```
plt.title('MSE Loss')
plt.plot(history['mae'], label='MAE (training data)',color='blue')
plt.plot(history['val_mae'], label='MAE (test data)',color='orange')
plt.ylabel('MAE Metric')
plt.xlabel('epoch')
plt.grid()
plt.legend(['MAE (training data)', 'MAE (validation data)'], loc='upper right')
plt.show()
```



Model Two : 2 layers with Sigmoid as of activation function

```
model = Sequential()
model.add(Dense(10, activation='sigmoid',input_shape=(15,),kernel_initializer='normal'))
model.add(Dense(1, kernel_initializer='normal'))
model.compile(loss='mean_squared_error', optimizer='adam',  metrics=['mse', 'mae'])
from keras import callbacks
earlystopping = callbacks.EarlyStopping(monitor ="val_loss", mode ="min", patience = 5, restore_best_weights = True,verbose=1)
trained_model = model.fit(X_train,y_train, batch_size=16, epochs=150, validation_split=0.15 ,callbacks =[earlystopping])
history = trained_model.history
```

evalute number of neurons and layers

```
# history
Y_pred = model.predict(X_test)
y_pred=np.ravel(Y_pred)
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('VarScore:',metrics.explained_variance_score(y_test,y_pred))
```

```
MAE: 557666.06
MSE: 621750500000.0
VarScore: 6.556510925292969e-07
```

- Model Three : 3 layers with ReLu as of activation function

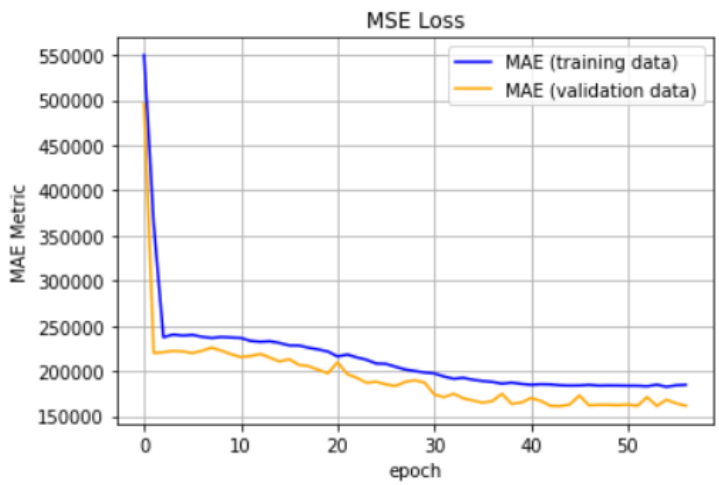
```
[ ] model = Sequential()
    model.add(Dense(10, activation='relu',input_shape=(15,),kernel_initializer='normal'))
    model.add(Dense(8,  activation='relu'))
    model.add(Dense(1, kernel_initializer='normal'))
    model.compile(loss='mean_squared_error', optimizer='adam',  metrics=['mse', 'mae'])
    from keras import callbacks
    earlystopping = callbacks.EarlyStopping(monitor ="val_loss", mode ="min", patience = 5, restore_best_weights = True,verbose=1)
    trained_model = model.fit(X_train,y_train, batch_size=16, epochs=150, validation_split=0.15 ,callbacks =[earlystopping])
    history = trained_model.history
```

evalute number of neurons and layers

```
[ ] # history
    Y_pred = model.predict(X_test)
    y_pred=np.ravel(Y_pred)
    from sklearn import metrics
    print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
    print('MSE:', metrics.mean_squared_error(y_test, y_pred))
    print('VarScore:',metrics.explained_variance_score(y_test,y_pred))
```

```
MAE: 178458.27
MSE: 255936950000.0
VarScore: 0.17681866884231567
```

```
plt.title('MSE Loss')
plt.plot(history['mae'], label='MAE (training data)',color='blue')
plt.plot(history['val_mae'], label='MAE (test data)',color='orange')
plt.ylabel('MAE Metric')
plt.xlabel('epoch')
plt.grid()
plt.legend(['MAE (training data)', 'MAE (validation data)'], loc='upper right')
plt.show()
```



Model Four : 3 layers with Sigmoid as of activation *function*

```
model = Sequential()
model.add(Dense(10, activation='sigmoid',input_shape=(15,),kernel_initializer='normal'))
model.add(Dense(8,  activation='sigmoid'))
model.add(Dense(1, kernel_initializer='normal'))
model.compile(loss='mean_squared_error', optimizer='adam',  metrics=['mse', 'mae'])
from keras import callbacks
earlystopping = callbacks.EarlyStopping(monitor ="val_loss", mode ="min", patience = 5, restore_best_weights = True,verbose=1)
trained_model = model.fit(X_train,y_train, batch_size=16, epochs=150, validation_split=0.15 ,callbacks =[earlystopping])
history = trained_model.history
```

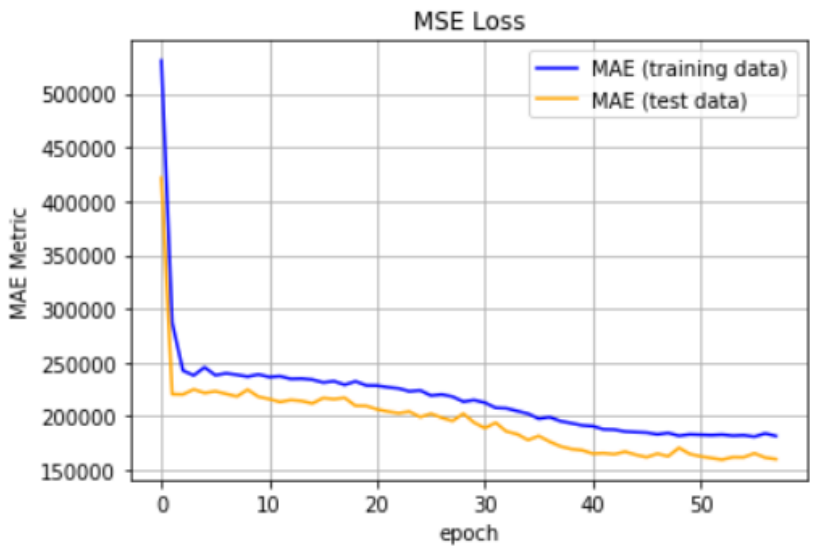
evalute number of neurons and layers

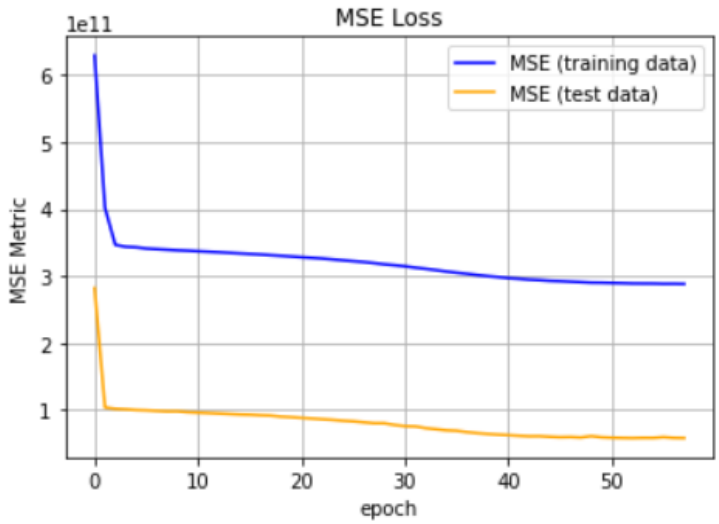
```
# history
Y_pred = model.predict(X_test)
y_pred=np.ravel(Y_pred)
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('VarScore:',metrics.explained_variance_score(y_test,y_pred))
```

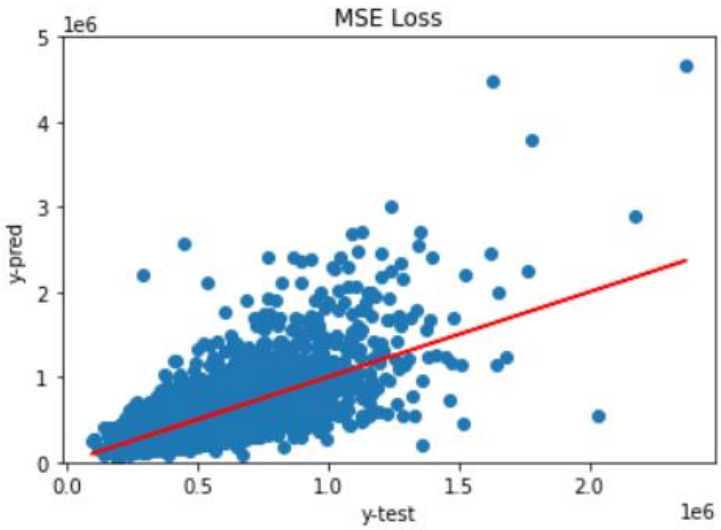
```
MAE: 557551.6
MSE: 621623100000.0
```

3.1.4 MSE Loss

Further you can see the result of the MSE Loss for mae and mse metrics.





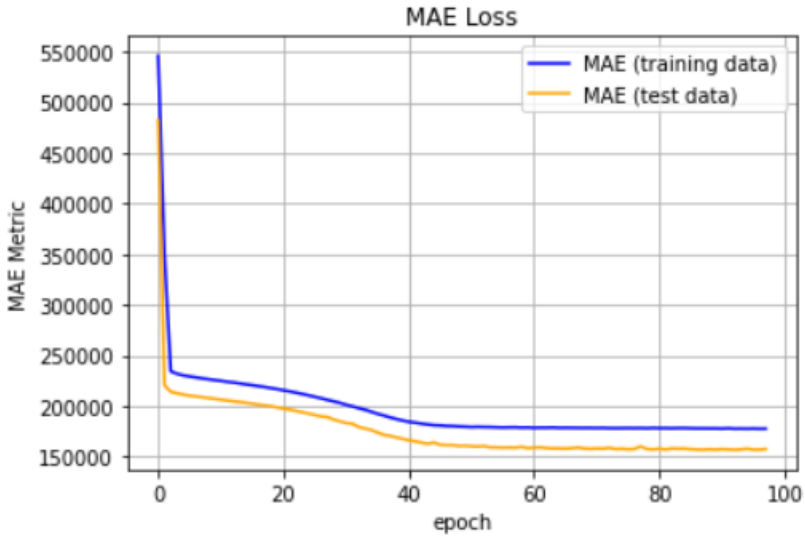


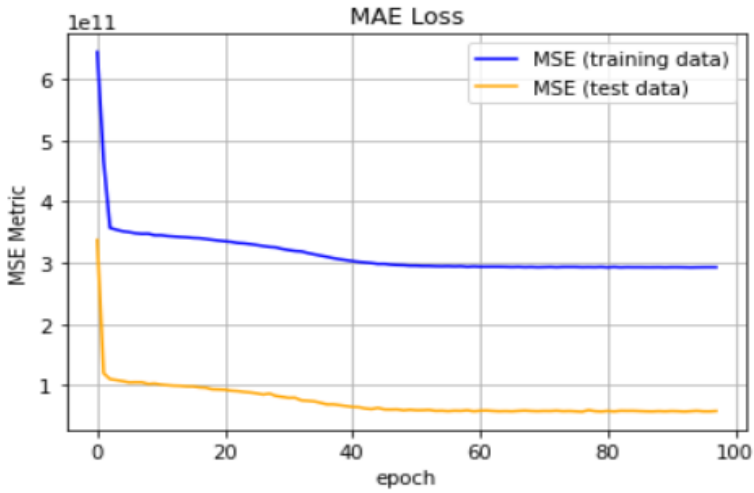
Number of optimal epochs :

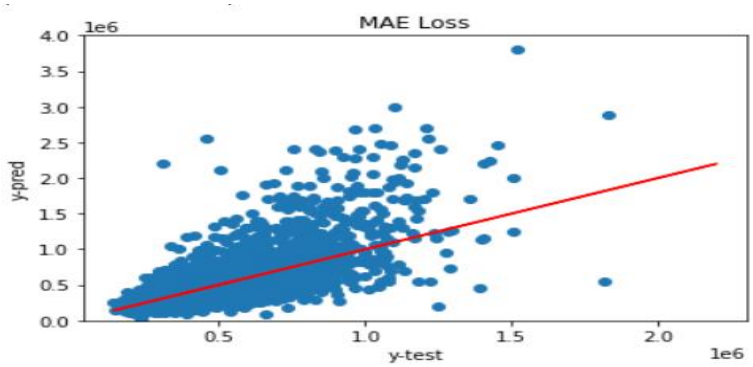
```
Epoch 58: early stopping
```

3.1.5 MAE Loss

Further you can see the result of the MAE Loss for mae and mse metrics.







Number of optimal epochs :

```
Epoch 98: early stopping
```

3.1.6 THEORETICAL APPROACH IN CASE OF MAE & MSE

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|real_i - predicted_i|$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(real_i - predicted_i)^2$$

3.1.7 BONUS SECTION

In Ridge algorithm with adding a new parameter to the loss equation with tend to minimize the summation of loss, and we tend to get a convergence with lower weight .

But in this case we observe that the result of ridge and regression are the same approximately.

Further you can see more detailed result.

Bonus Section

```
[ ] from sklearn.linear_model import LinearRegression
    from sklearn.linear_model import Ridge
```

Linear Regression

```
[ ] regressor = LinearRegression()
    regressor.fit(X_train, y_train)
```

```
LinearRegression()
```

```
#compare actual output values with predicted values
y_pred = regressor.predict(X_test)

# evaluate the performance of the algorithm (MAE - MSE - RMSE)
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('VarScore:',metrics.explained_variance_score(y_test,y_pred))
```

```
MAE: 162243.2
MSE: 244478700000.0
VarScore: 0.2132859230041504
```

Ridge Regression

```
[ ] # define model
    Ridge_model = Ridge(alpha=0.9)
    Ridge_model.fit(X_train, y_train)
```

```
Ridge(alpha=0.9)
```

```
#compare actual output values with predicted values
y_pred = Ridge_model.predict(X_test)

# evaluate the performance of the algorithm (MAE - MSE - RMSE)
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('VarScore:',metrics.explained_variance_score(y_test,y_pred))
```

```
MAE: 162259.86
MSE: 244479670000.0
VarScore: 0.21328282356262207
```

4 QUESTION #3

4.1 REDUCING DIMENSIONS

In this part we intend to implement reduction dimensions using PCA & Auto-encoder to be used on Cifar10 dataset.

4.1.1 PCA

In short, PCA is a dimensionality reduction technique that transforms a set of features in a dataset into a smaller number of features called principal components while at the same time trying to retain as much information in the original dataset as possible

Assume x denotes an n-dimensional point of the input space: $x^T=[1,x_1,x_2,..,x_n]$ (bias has been augmented)

There will be a linear correlation among different input dimensions, if for all samples of x there is a non-zero real vector(the normal vector), $a^T=[a_0,a_1,..,a_n]$, in order that:

$$a^T x = 0, \quad x \in \{x^i\}_{i=1}^m$$

One can show for *r* independent linear correlations, "r" Eigen values of the "Auto Correlation Matrix" (R) are zero and their corresponding Eigen vectors reveal "r" independent linear correlations.

$$R = \sum_{i=1}^{m} x^i x^{iT} = V\Lambda V^T$$

$$\Lambda = \begin{bmatrix} \Lambda^1 & 0 \\ 0 & \Lambda^2 \end{bmatrix} \quad \Lambda^1 = diag([\lambda_1,..,\lambda_r]) = \mathbf{0} \quad \Lambda^2 = diag([\lambda_{r+1}....,\lambda_n])$$

$$V = \left[\underbrace{v_1..v_r}_{V_1}\ \underbrace{v_{r+1}...v_n}_{V_2}\right] = [V_1\ V_2]$$

r independent Correlations: $v_j^T x = 0 (j \epsilon \{1,\dots,r\})$

Dimensionality Reduction and Data Retrieving for Linear Correlations

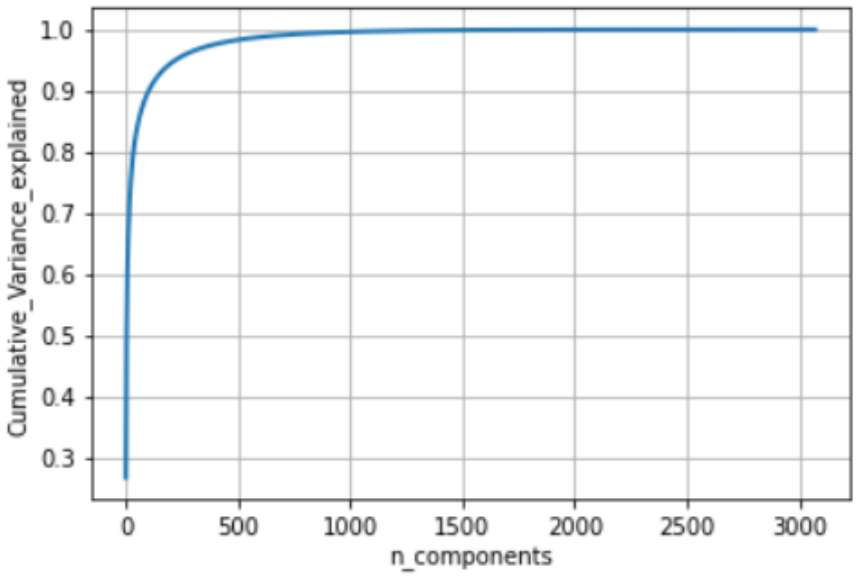
$$V^T x = \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} x = \begin{bmatrix} V_1^T x \\ V_2^T x \end{bmatrix} = \begin{bmatrix} 0_{r\times 1} \\ z_{(n-r)\times 1} \end{bmatrix}$$

Nullity part which will be removed.

Informative part which forms the new space

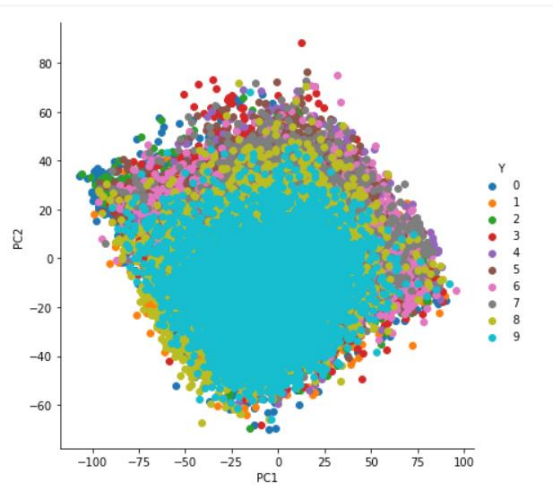
$$x \epsilon R^n \xrightarrow{Dimensionality\ Reduction} z(x) = Tr(x) = V_2^T x \quad z \epsilon R^{n-r}$$

$$z \epsilon R^{n-r} \xrightarrow{Data\ retrieving} x = Tr^{-1}(z) = V \begin{bmatrix} 0_{r\times 1} \\ z \end{bmatrix} = VV^T x = x\ \epsilon R^{n-r}$$




```
### if n_components= 645,   variance=99.000000
### if n_components= 440,   variance=98.000000
### if n_components= 334,   variance=97.000000
### if n_components= 265,   variance=96.000000
### if n_components= 218,   variance=95.000000
### if n_components= 182,   variance=94.000000
### if n_components= 155,   variance=93.000000
### if n_components= 133,   variance=92.000000
### if n_components= 115,   variance=91.000000
### if n_components= 101,   variance=90.000000
```

So to protect 90 percent of information we hold 102 components.



As you can see the above figure the two most priority in case of information have been plotted together.

And further we have trained the data on PCA.

```
model = Sequential()
model.add(Dense(512,  activation='relu', input_shape=(646,)))
model.add(Dense(128,  activation='relu'))
model.add(Dense(10,    activation='softmax'))
model.compile(loss='mean_squared_error', optimizer = Adam())
trained_model = model.fit(X_train,y_train, batch_size=16, epochs=200, verbose=1, validation_data=(X_test,y_test))
```

```
Epoch 1/20
2700/2700 [==============================] - 12s 3ms/step - loss: 1.7916 - val_loss: 1.5963
Epoch 2/20
2700/2700 [==============================] - 8s 3ms/step - loss: 1.5022 - val_loss: 1.4930
Epoch 3/20
2700/2700 [==============================] - 9s 3ms/step - loss: 1.3930 - val_loss: 1.4763
Epoch 4/20
2700/2700 [==============================] - 9s 3ms/step - loss: 1.3200 - val_loss: 1.4434
Epoch 5/20
2700/2700 [==============================] - 9s 4ms/step - loss: 1.2641 - val_loss: 1.4339
Epoch 6/20
2700/2700 [==============================] - 9s 3ms/step - loss: 1.2185 - val_loss: 1.4373
Epoch 7/20
2700/2700 [==============================] - 8s 3ms/step - loss: 1.1805 - val_loss: 1.4540
Epoch 8/20
2700/2700 [==============================] - 8s 3ms/step - loss: 1.1481 - val_loss: 1.4469
Epoch 9/20
2700/2700 [==============================] - 8s 3ms/step - loss: 1.1232 - val_loss: 1.4536
Epoch 10/20
2700/2700 [==============================] - 8s 3ms/step - loss: 1.0981 - val_loss: 1.4720
Epoch 11/20
2700/2700 [==============================] - 10s 4ms/step - loss: 1.0743 - val_loss: 1.4680
Epoch 12/20
2700/2700 [==============================] - 8s 3ms/step - loss: 1.0562 - val_loss: 1.5060
Epoch 13/20
2700/2700 [==============================] - 8s 3ms/step - loss: 1.0379 - val_loss: 1.5340
Epoch 14/20
2700/2700 [==============================] - 8s 3ms/step - loss: 1.0183 - val_loss: 1.5420
Epoch 15/20
2700/2700 [==============================] - 8s 3ms/step - loss: 1.0073 - val_loss: 1.5211
```

```
Epoch 16/20
2700/2700 [==============================] - 8s 3ms/step - loss: 0.9890 - val_loss: 1.5260
Epoch 17/20
2700/2700 [==============================] - 8s 3ms/step - loss: 0.9753 - val_loss: 1.5686
Epoch 18/20
2700/2700 [==============================] - 8s 3ms/step - loss: 0.9678 - val_loss: 1.5525
Epoch 19/20
2700/2700 [==============================] - 8s 3ms/step - loss: 0.9552 - val_loss: 1.5400
Epoch 20/20
2700/2700 [==============================] - 8s 3ms/step - loss: 0.9411 - val_loss: 1.6313
```

The performance of classifying using PCA:

```
[27] predicted = model.predict(X_test)
     predicted=np.argmax(predicted, axis=1)
     Y_test=np.argmax(Y_test, axis=1)

     accuracy_score(predicted,Y_test)*100

     49.016666666666666
```

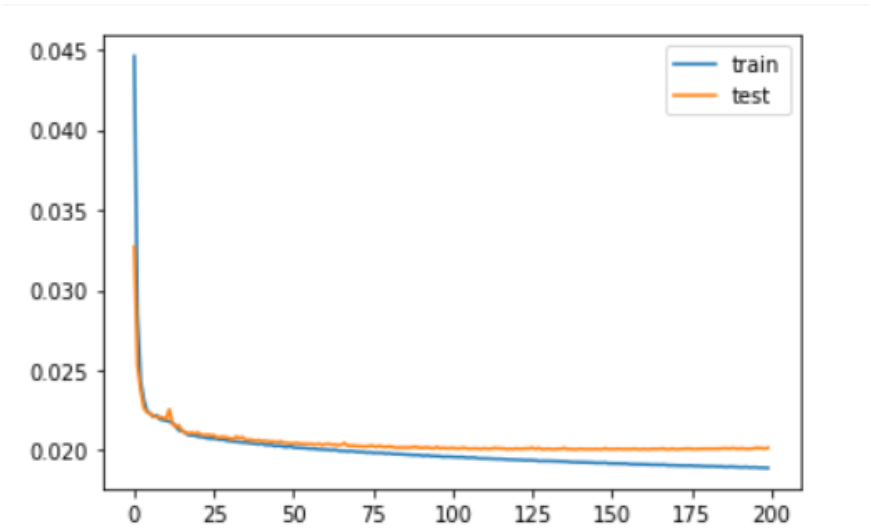
As we can see the result of the training on PCA algorithm is pretty much better and so it has been implemented correctly.

4.1.2 AUTO-ENCODER

in this part I have implemented a auro-encoder using neural nets and further you can see more detailed information about that.

```
] autoencoder = Sequential()
  autoencoder.add(Dense(512,  activation='elu', input_shape=(3072,)))
  autoencoder.add(Dense(128,  activation='elu'))
  autoencoder.add(Dense(10,    activation='linear', name="bottleneck"))
  autoencoder.add(Dense(128,  activation='elu'))
  autoencoder.add(Dense(512,  activation='elu'))
  autoencoder.add(Dense(3072,  activation='sigmoid'))
  autoencoder.compile(loss='mean_squared_error', optimizer = Adam())
  trained_model = autoencoder.fit(X_train,X_train, batch_size=1024, epochs=200, verbose=1, validation_data=(X_test,X_test))
  encoder = Model(autoencoder.input, autoencoder.get_layer('bottleneck').output)
  encoded_data = encoder.predict(X_train)  # bottleneck representation
  decoded_output = autoencoder.predict(X_train)        # reconstruction
  encoding_dim = 10

  # return the decoder
  encoded_input = Input(shape=(encoding_dim,))
  decoder = autoencoder.layers[-3](encoded_input)
  decoder = autoencoder.layers[-2](decoder)
  decoder = autoencoder.layers[-1](decoder)
  decoder = Model(encoded_input, decoder)
```



Train-result

After training the encoder we can use that to transform train and test to the reduction space and then classify dataset using this approach.

As you see further the accuracy is about 33% percent which is lower than in comparison to PCA so it is not a ideal choice for our work.

```
[2 8 8 ... 5 5 7]
[[3]
 [8]
 [8]
 ...
 [5]
 [1]
 [7]]
0.3322
```

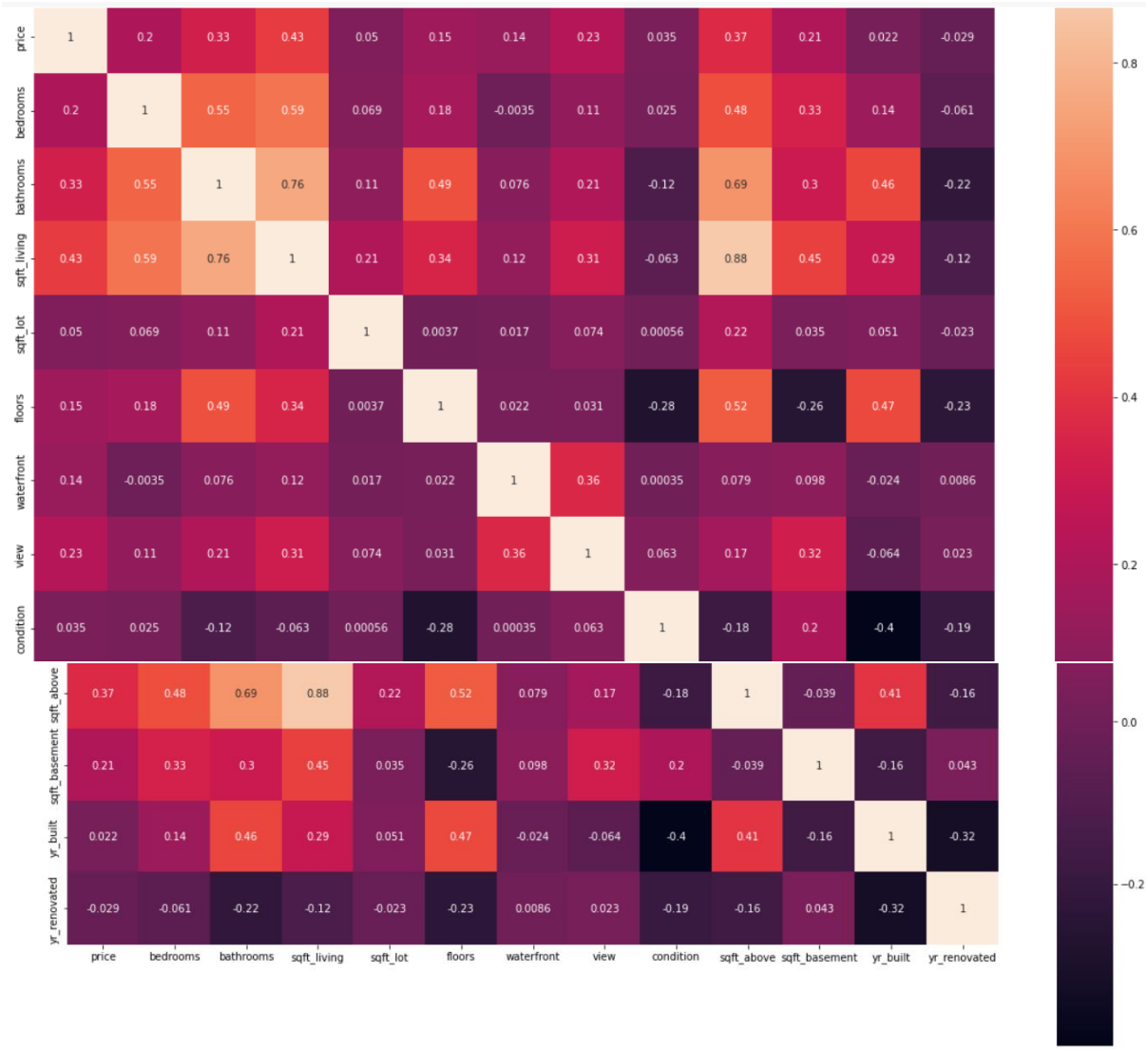
Comparing two approach :

We can see that in PCA we achieve a better performance because the image has been reduced as well as it could be.

But in this case auto-encoder doesn't work really good and it can be made by redundancy and high correlation between images.

4.1.3 COVARIANCE MATRIX

Further the covariance matrix have been plotted:

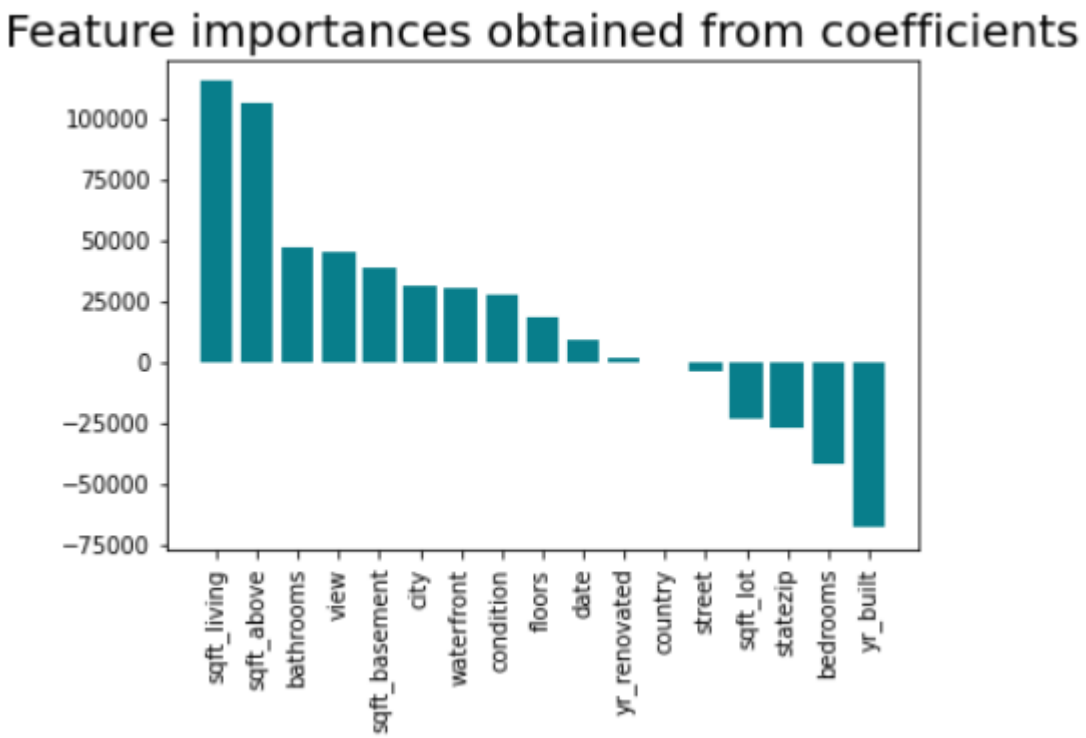


As you can see the main Diameter get a 1 value always and this is because the correlation between two element tends to be maxsimum and also the every element outside main diameter show the correlation between row i and column j and when this value become larger it shows that we may need to feature redunction.

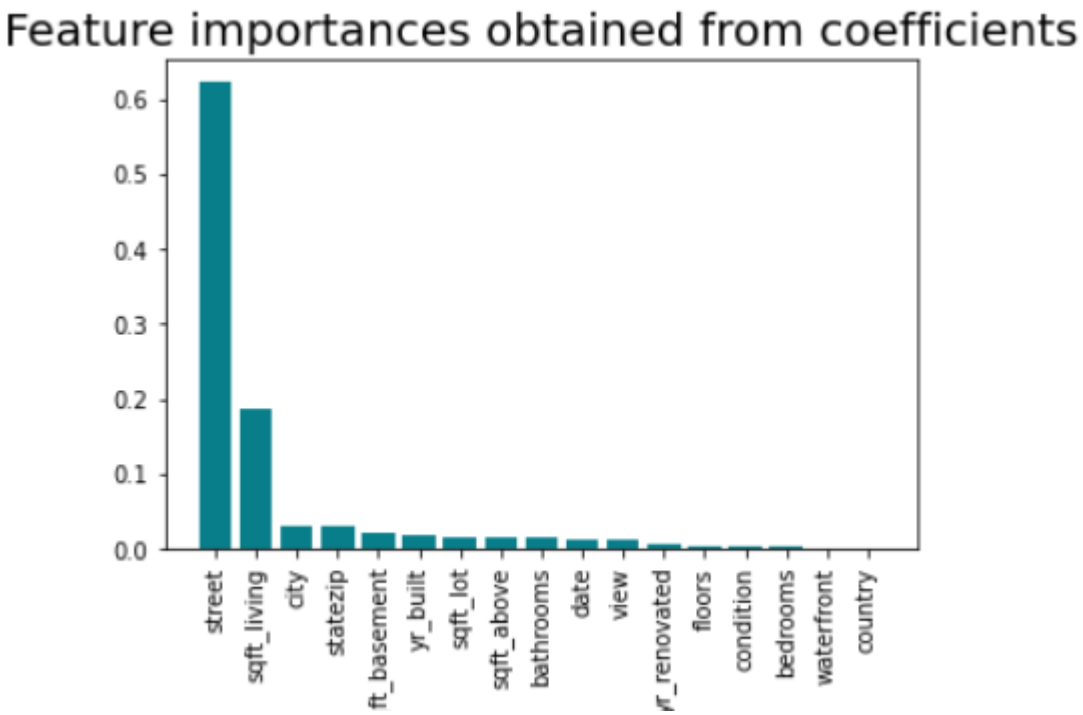
4.1.4 FEATURE IMPORTANCE:

As you can see further we have plotted the barplot using decision tree and linear regression :

Linear regression:



Decision tree :



5 ACKNOWLEDGEMENT

I am really grateful for Mr MohammadHossein Vaeedi (810197605) because in some part of this project we had a nice and effective discussion which highly helped us to have a better analyse and also have more adequate results! (Note that we only talked about ideas behind different problems.)

Afterwards, I am thankful to all of course teaching assistants: MojtabaAmiri(MojtabaAmiri@ut.ac.ir) and AliAzizi(aziziali.9473@gmail.com) who designed this project with high quality.

6 REFERENCES

[1] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html
[2] https://towardsdatascience.com/mercari-price-suggestion-97ff15840dbd
[3]https://towardsdatascience.com/deep-neural-multilayer-perceptron-mlp-with-scikit-learn-2698e77155e
[4]https://medium.com/data-science-bootcamp/multilayer-perceptron-mlp-vs-convolutional-neural-network-in-deep-learning-c890f487a8f1
[5] https://medium.com/@mygreatlearning/step-by-step-regression-analysis-f7e3e3ebf296
[6] https://medium.com/analytics-vidhya/basic-regression-models-5153454fe62f
[7] https://builtin.com/data-science/step-step-explanation-principal-component-analysis
[8]https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c
[9] https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798