Deep Learning

Assignment 2 Part 2

Pramil Panjawani (PhD19008)

Reshan Faraz (PhD19006)

Date : 26-Feb-2021

Q1) Best optimizer find the ReLU function

1) **Optimizer**

Optimizer	**Train accuracy**
Standard GD	0.7745
GD + Momentum	0.9027
NAG	0.913
Adagrad	0.8351
RMSprop	0.778
ADAM	0.7978

2) **Initialization Technique**

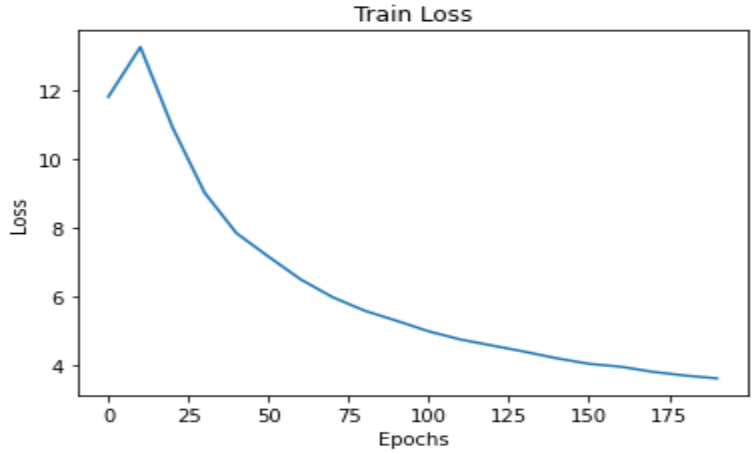
	Train accuracy
He	0.938
Xavier	0.9374

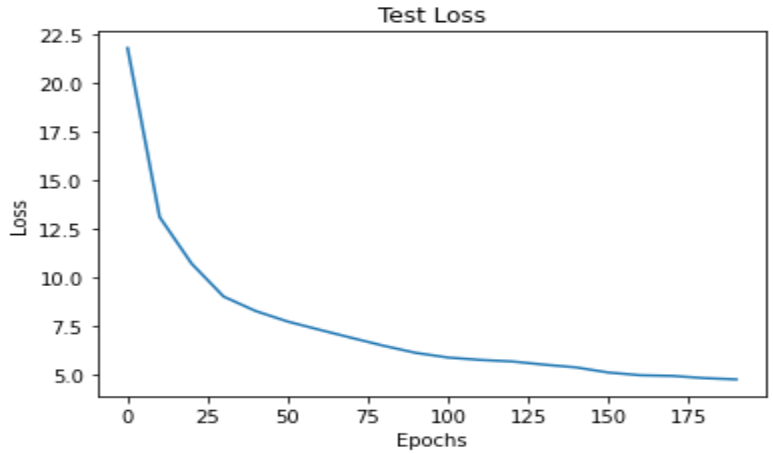
3) **Regularization**

	Train accuracy
L1	0.9367
L2	0.9384
Dropout	0.9416

4) Train and Test loss vs Epochs

i) Gradient Descent

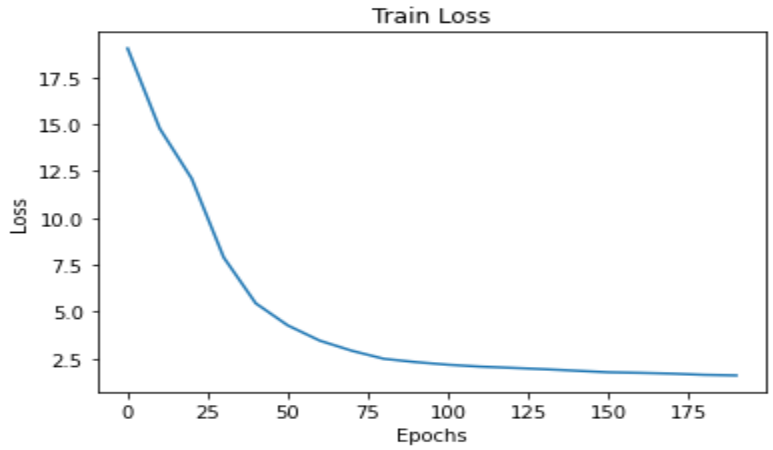


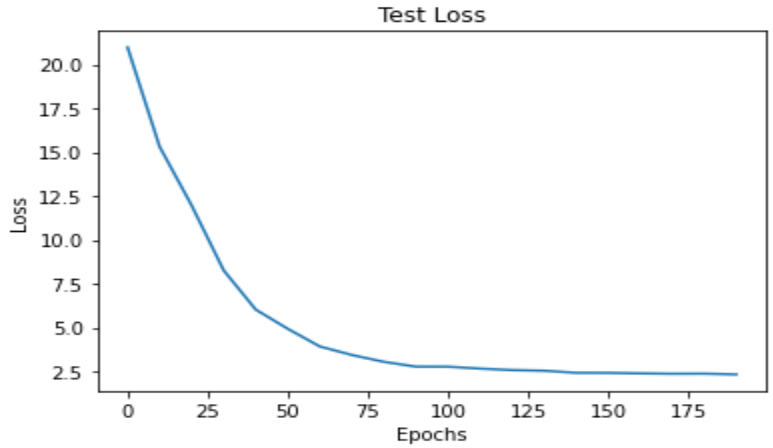


Confusion Matrix

```
array([[910,   0,   9,   5,   7,  38,  10,   6,  14,   1],
       [  0, 943,   7,  10,   0,   8,   4,   6,  19,   3],
       [ 18,  53, 722,  32,  20,   5,  89,   9,  35,  17],
       [ 11,  23,  44, 696,   7,  62,   4,  27,  99,  27],
       [  6,   2,  16,   4, 752,  12,  13,  14,  32, 149],
       [ 55,  16,  11,  76,  22, 628,  39,   3, 134,  16],
       [ 23,   4,  65,   0,  18,  18, 843,   2,  27,   0],
       [  6,  19,  15,  24,  30,   1,   0, 842,   2,  61],
       [  9,  35,  32,  88,  23,  80,  15,   5, 684,  29],
       [ 21,   2,   7,  46, 141,  17,   3,  77,  25, 661]])
```

ii) Gradient Descent + Momentum

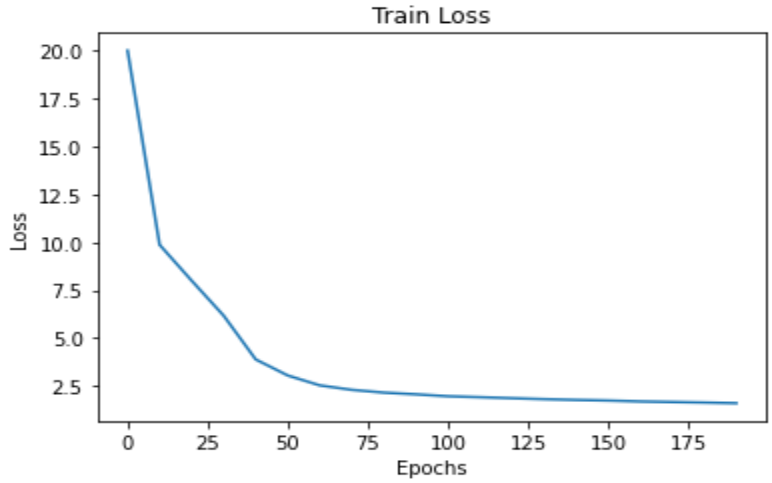


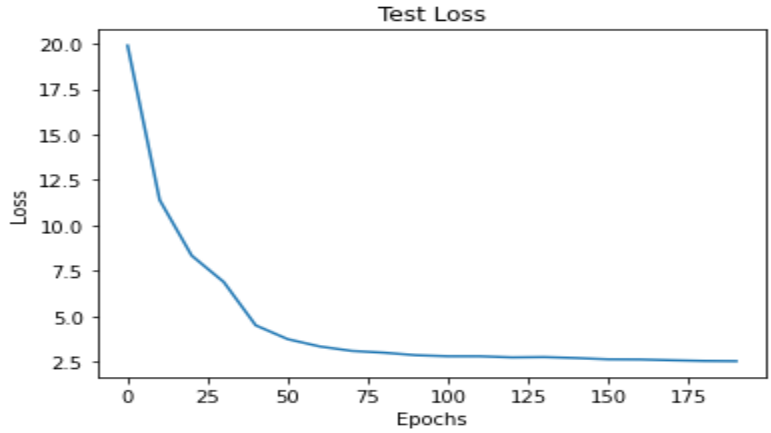


Confusion Matrix

```
array([[963,   0,   2,   4,   4,  11,   6,   3,   7,   0],
       [  0, 963,   7,   3,   1,   7,   0,   2,  14,   3],
       [  8,  16, 870,  19,   7,   5,  26,   7,  34,   8],
       [  4,  10,  24, 867,   3,  44,   0,  18,  20,  10],
       [  0,   5,   6,   0, 912,   3,  23,   3,   5,  43],
       [ 15,   5,   7,  41,   9, 865,  24,   1,  23,  10],
       [ 11,   2,  17,   1,  17,  11, 928,   0,  11,   2],
       [  1,   6,  11,   8,  16,   3,   0, 918,   2,  35],
       [  3,  15,  18,  28,   3,  43,  21,   2, 847,  20],
       [ 11,   5,   4,  11,  35,   5,   2,  26,   7, 894]])
```

iii) NAG

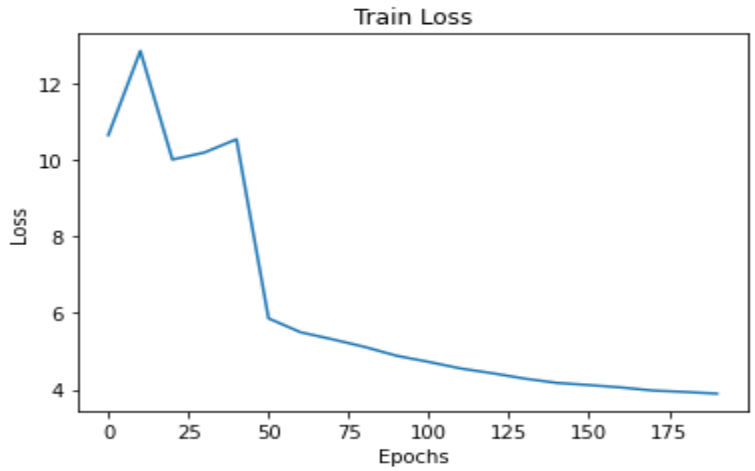


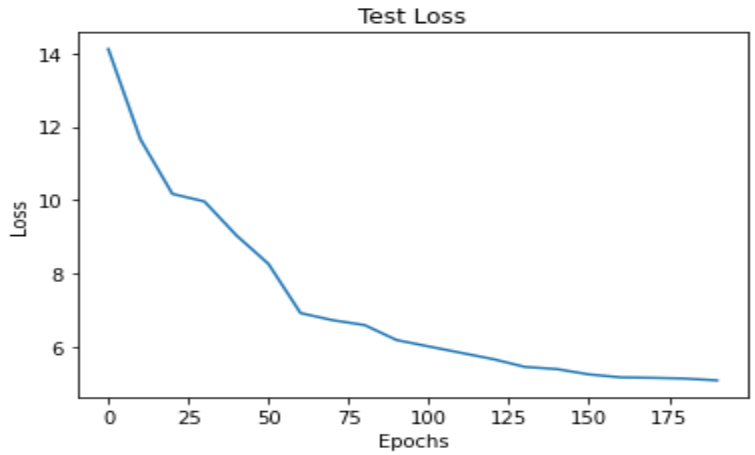


Confusion Matrix

```
array([[967,   0,   4,   2,   2,   7,   6,   5,   7,   0],
       [  0, 966,   5,   7,   0,   5,   0,   3,  13,   1],
       [  8,  18, 872,  20,  13,   3,  20,  10,  31,   5],
       [  1,   7,  30, 858,   2,  53,   4,  17,  21,   7],
       [  1,   2,   8,   0, 895,   3,  11,   1,  13,  66],
       [ 17,   2,   3,  46,  11, 847,  21,   0,  43,  10],
       [  9,   2,  12,   1,  13,  12, 940,   0,  11,   0],
       [  2,  10,   9,   5,   8,   2,   0, 916,   4,  44],
       [  6,  23,  18,  20,   4,  46,  19,   5, 844,  15],
       [  9,   1,   2,   8,  33,   5,   2,  39,  15, 886]])
```

iv) Adagrad

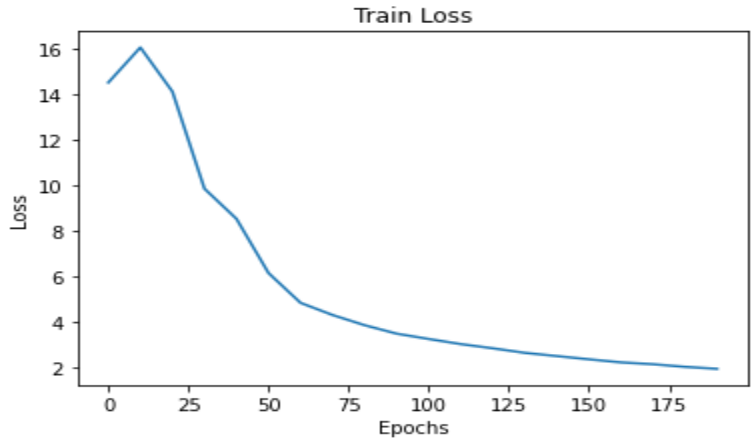


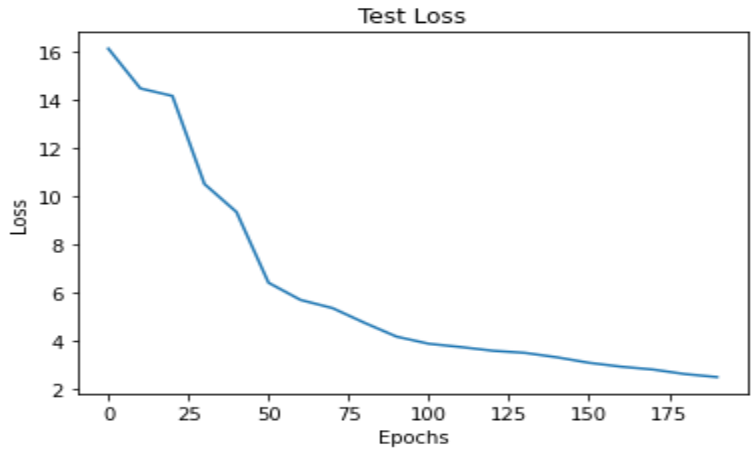


Confusion Matrix

```
array([[912,   0,   1,  42,   1,  29,   2,  12,   1,   0],
       [  0, 947,  25,   2,   2,   0,   0,   2,  19,   3],
       [  7,  55, 668, 119,  13,  24,  59,   5,  49,   1],
       [ 13,  13,  43, 689,   3,  79,   3,  31, 119,   7],
       [  1,  27,   5,  15, 820,   1,   5,   8,   4, 114],
       [107,   9,  54, 410,   5, 343,  28,  36,   7,   1],
       [  5,  23,  59,   7,   4,  22, 871,   9,   0,   0],
       [ 13,   7,   3,  12,  13,   0,   0, 799,  40, 113],
       [  5,  61,  73, 229,   5,  12,   5,  10, 575,  25],
       [  5,   5,   0,  18,  96,   0,   0, 122,   9, 745]])
```

v) RMSprop

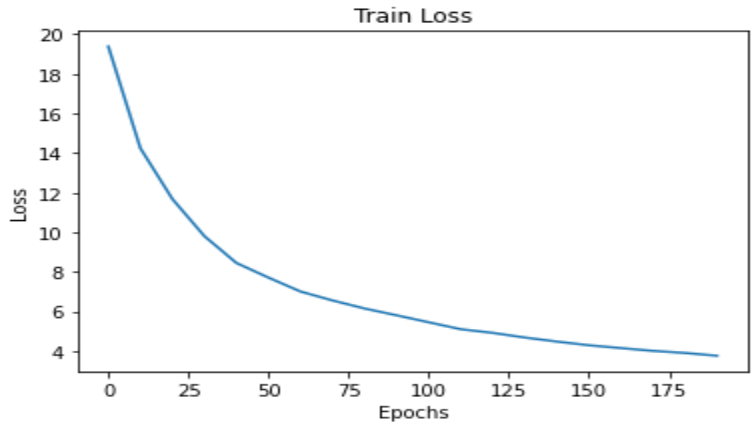


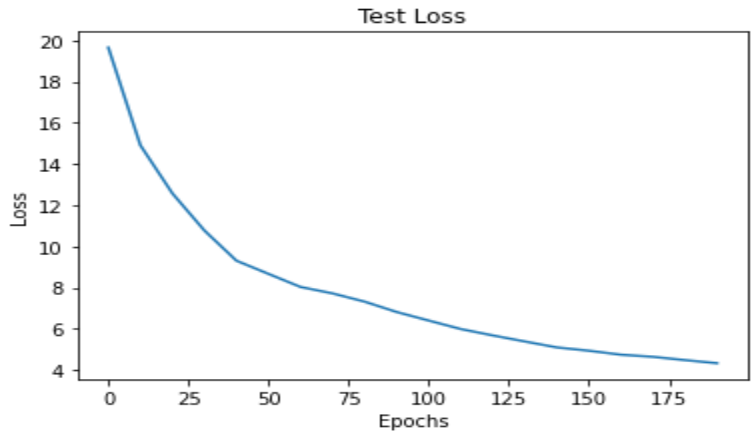



```
# print(np.mean(y_pred == y_test))
print("Accuracy",np.mean(y_pred == y_test)*100)

print("Confusion Matrix = \n",confusion_matrix(y_test, y_pred))
```

```
Accuracy 77.7
Confusion Matrix =
 [[167   0   1   2   0   8  12   3   5   2]
 [  0 186   1   1   0   6   3   1   2   0]
 [  0   1 158   4   0   1  13   8  12   3]
 [  2   0  11 131   0  27   6  10  10   3]
 [  0   0   1   0 131   0  14   2   3  49]
 [  8   0   6   7   7 143   3   2  21   3]
 [  4   3   7   0   6   5 169   1   5   0]
 [  0   7   8   1   1   1   1 166   2  13]
 [  4   4   9   4   6  12   3   9 137  12]
 [  1   1   0   3   5   2   3  13   6 166]]
```

vi) Adam





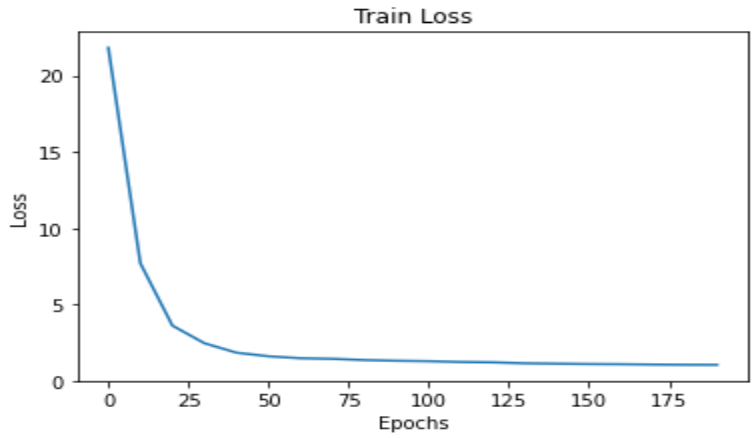

```
print("Accuracy",np.mean(y_pred == y_test)*100)

print("Confusion Matrix = \n",confusion_matrix(y_test, y_pred))
```

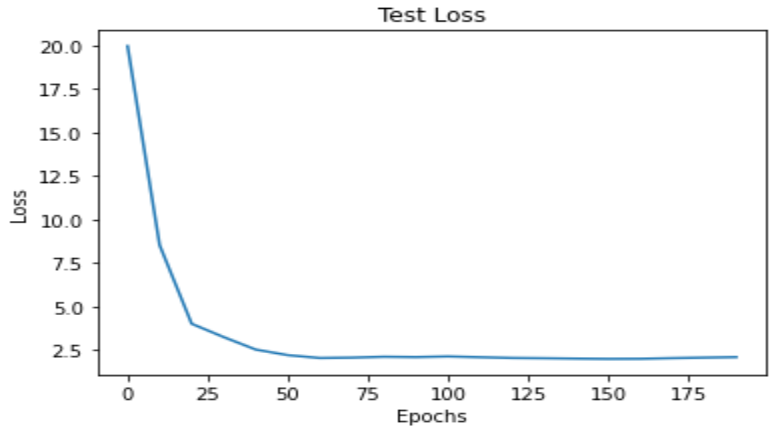
```
Accuracy 79.4
Confusion Matrix =
 [[183   0   2   1   1   8   3   1   1   0]
 [  0 183   3   4   1   2   0   0   6   1]
 [  3   1 163   9   1   2   3   8   7   3]
 [  8   1  13 137   1  21   0  10   5   4]
 [  1   2   2   1 177   1   5   3   1   7]
 [  6   0   0   6   2 171   1   6   6   2]
 [ 12   2   7   1   7   9 152   5   5   0]
 [  0   4   5   4  10   0   1 159   3  14]
 [  5   0   7   8  11  24   4   3 125  13]
 [  1   1   1   2  30  13   1  11   2 138]]
```

Q2) Results of Weight Initialization Techniques:

We observe NAG with ReLU gives the best accuracy

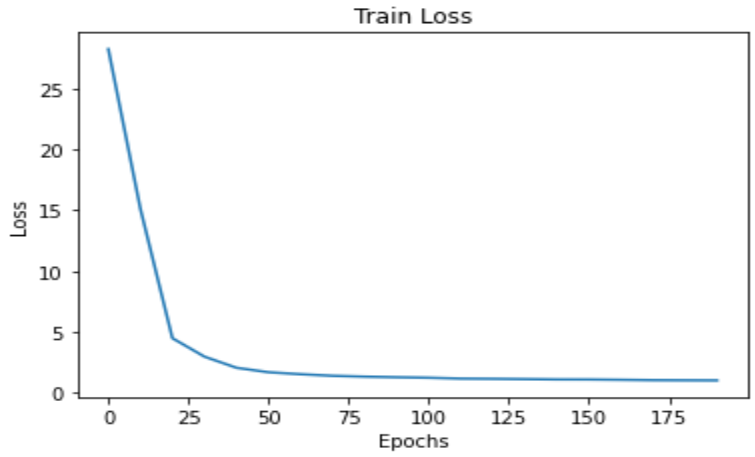
1) He initialization

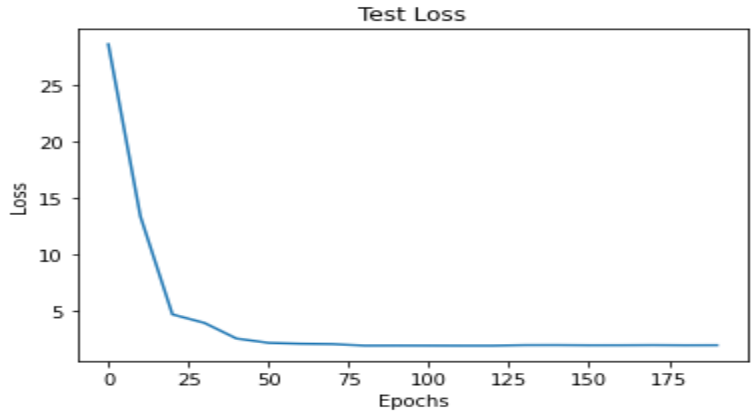





```
Confusion Matrix
array([[979,   0,   1,   3,   1,   3,   4,   2,   6,   1],
       [  0, 974,   4,   1,   3,   6,   0,   1,   9,   2],
       [  4,  10, 912,  10,   8,   6,   8,   9,  26,   7],
       [  1,   7,  25, 907,   0,  34,   2,   7,   8,   9],
       [  1,   4,   7,   0, 951,   0,   9,   3,   6,  19],
       [  7,   5,   4,  33,  11, 910,  13,   0,  13,   4],
       [  7,   2,   6,   0,   5,  10, 963,   0,   5,   2],
       [  1,   4,   8,   3,  12,   0,   0, 950,   3,  19],
       [  4,  12,  11,  15,   6,  22,  14,   3, 905,   8],
       [  7,   2,   2,  13,  14,   1,   0,  26,   6, 929]])
```

2) Xavier Initialization



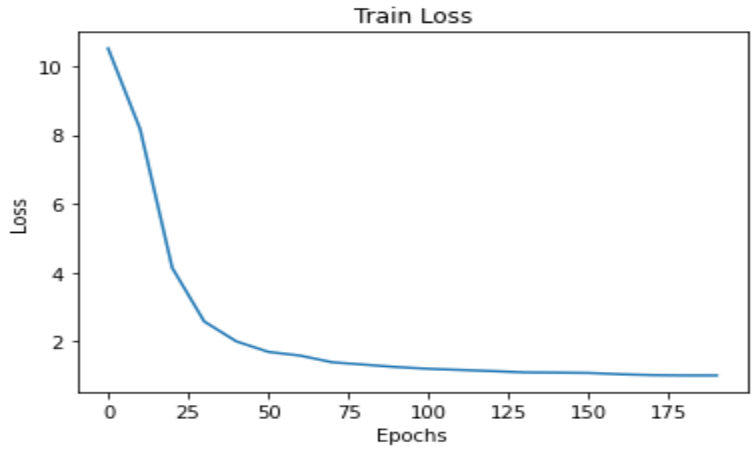


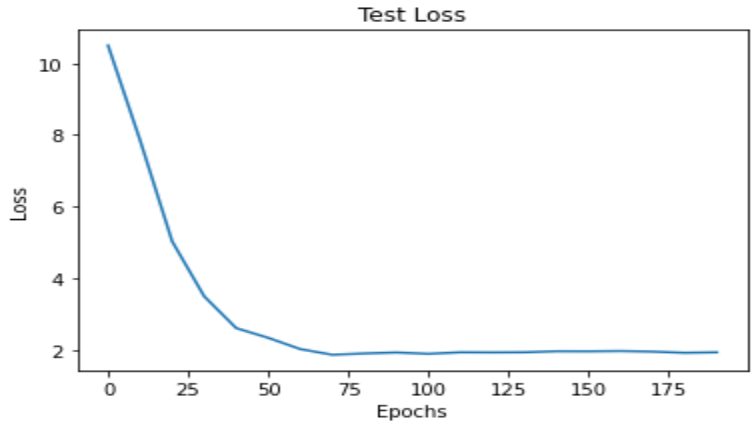

```
Confusion Matrix
array([[985,   0,   2,   1,   2,   2,   3,   1,   4,   0],
       [  0, 973,   4,   1,   2,   8,   0,   3,   7,   2],
       [  3,   9, 912,   8,   7,   8,  12,  10,  25,   6],
       [  1,   6,  23, 907,   1,  34,   2,   9,   8,   9],
       [  1,   6,   8,   0, 940,   2,   8,   1,   7,  27],
       [  8,   4,   5,  29,  11, 905,  17,   0,  16,   5],
       [  5,   2,   4,   0,   6,  10, 967,   0,   5,   1],
       [  3,   4,   8,   3,  12,   0,   0, 953,   1,  16],
       [  4,  13,  11,  15,   6,  24,  11,   3, 905,   8],
       [  6,   2,   4,  11,  15,   2,   0,  25,   8, 927]])
```

Q3) Results of Regularization Techniques:

We observe both He and Xavier Initialization giving similar results, but since we are using ReLU activation we will proceed with

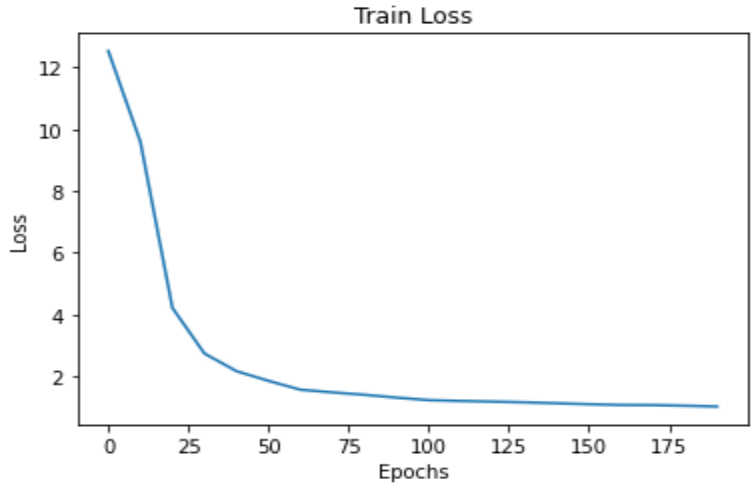
1) **L1**

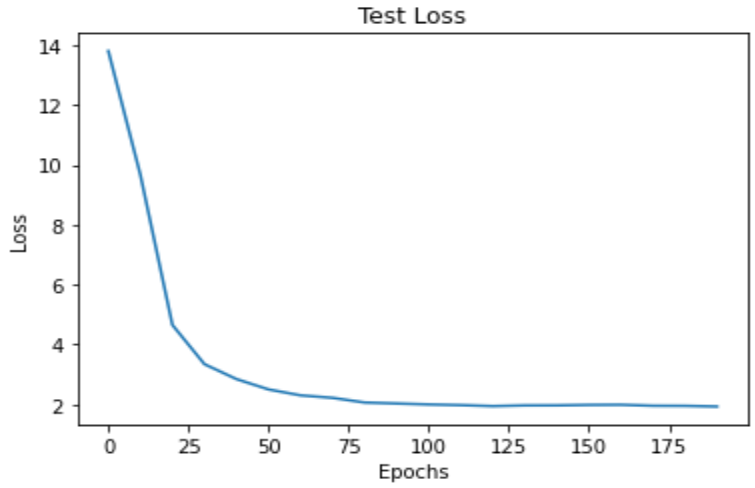





```
Confusion Matrix
array([[982,   0,   1,   3,   2,   2,   4,   2,   4,   0],
       [  0, 975,   4,   1,   2,   6,   0,   1,   9,   2],
       [  4,   9, 921,   8,   9,   7,  11,   8,  18,   5],
       [  2,   7,  21, 907,   1,  35,   1,   7,   9,  10],
       [  0,   3,   9,   0, 946,   0,   9,   2,   7,  24],
       [ 11,   3,   6,  31,  12, 904,  11,   0,  15,   7],
       [  4,   2,   7,   0,   5,  10, 966,   1,   5,   0],
       [  3,   5,   8,   4,  11,   0,   0, 936,   2,  31],
       [  5,  14,   8,  13,   6,  21,  13,   3, 909,   8],
       [  5,   2,   5,  12,  17,   1,   0,  32,   5, 921]])
```

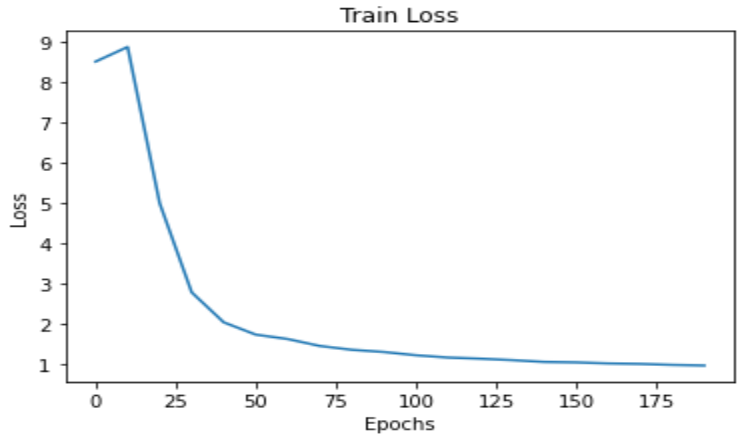
2) L2

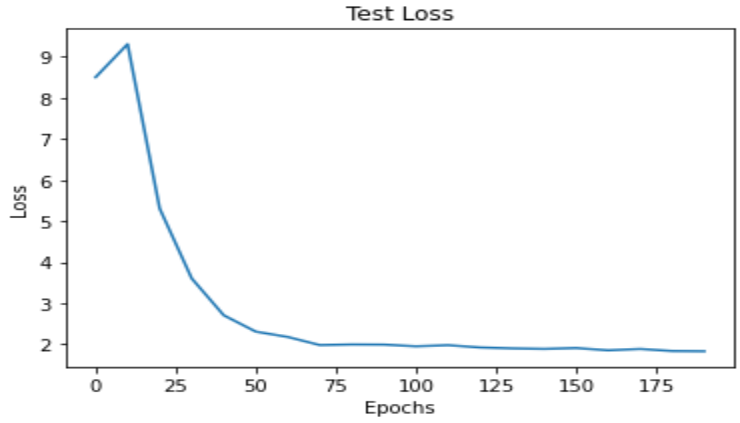





```
Confusion Matrix
array([[982,   0,   1,   3,   2,   3,   3,   2,   4,   0],
       [  0, 966,   6,   2,   3,   7,   0,   2,  12,   2],
       [  5,  11, 914,   9,   7,   5,  10,  10,  24,   5],
       [  0,   9,  23, 908,   1,  33,   2,   7,   6,  11],
       [  1,   5,   7,   1, 945,   1,   9,   4,   6,  21],
       [ 11,   5,   5,  25,  11, 909,  12,   0,  18,   4],
       [  5,   2,   4,   0,   5,  10, 969,   0,   5,   0],
       [  1,   4,   7,   2,  12,   0,   0, 952,   2,  20],
       [  5,  13,  11,  12,   7,  17,  13,   4, 908,  10],
       [  7,   2,   4,  12,  10,   3,   0,  25,   6, 931]])
```

3) Dropout






```
Confusion Matrix
array([[984,   0,   2,   1,   3,   1,   1,   3,   5,   0],
       [  0, 978,   4,   3,   2,   5,   0,   0,   6,   2],
       [  3,  11, 916,  13,   9,   8,   7,   6,  20,   7],
       [  2,   7,  23, 901,   2,  35,   1,  10,   8,  11],
       [  1,   5,   6,   0, 945,   0,   8,   1,   7,  27],
       [  8,   4,   5,  29,  10, 908,  12,   1,  18,   5],
       [  5,   2,   3,   0,   4,   7, 972,   1,   5,   1],
       [  3,   6,   9,   4,  10,   0,   0, 954,   1,  13],
       [  4,  14,  10,  15,   7,  12,   9,   1, 919,   9],
       [  6,   1,   1,  12,  13,   1,   0,  22,   5, 939]])
```

Conclusion

We observe that for optimizers Nesterov's gradient descent performs best for both test and train accuracy and takes somewhere around 25-30 epochs to converge. Followed by gradient descent with momentum and adagrad. Adam and RMSprop don't perform as well but still outperform standard gradient descent.

For weight initialization, both He and Xavier perform equally well for all practical purposes. Both show a slight improvement over random initialization.

Dropout regularization performs slightly better than L1 and L2 regularisation, both of which perform equally well.

With standard training, it takes up to 500 epochs to reach 85% accuracy, which we outperform significantly in only 25-30 epochs when using optimizers and regularization.