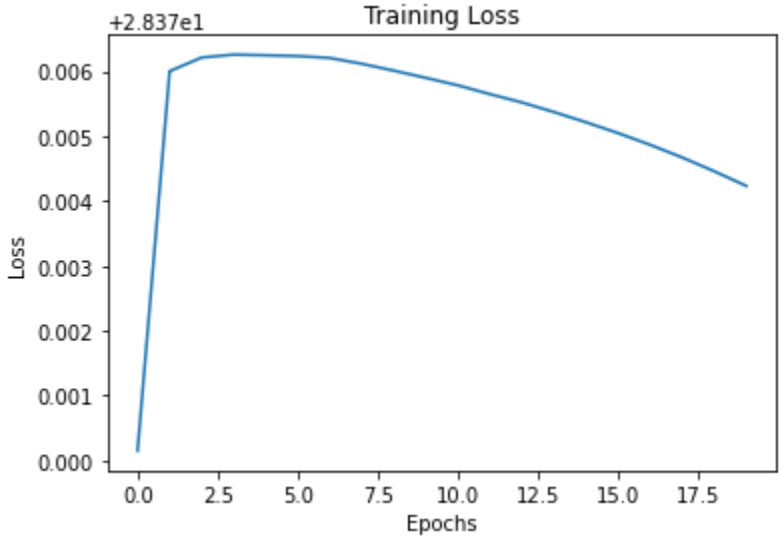
Report

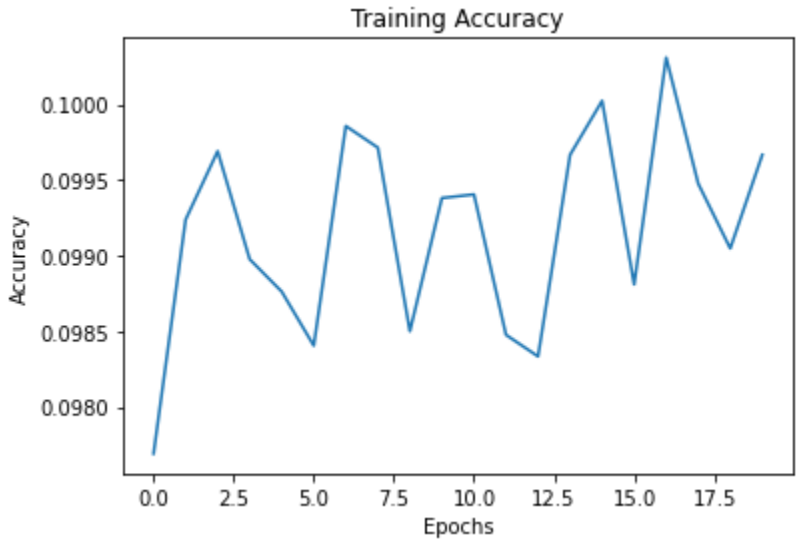
1. Optimizers

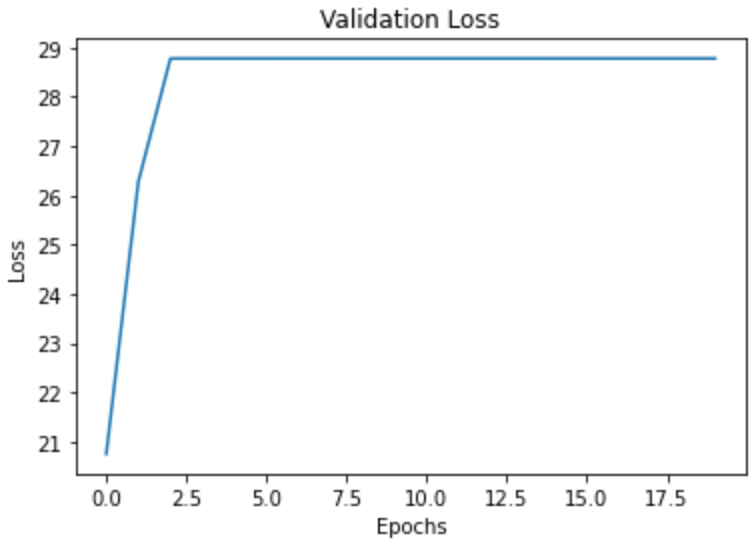
(Results for Epochs = 20, Alpha = 0.4, Beta = 0.4, Gamma = 0.4, Epsilon = 0.0001)

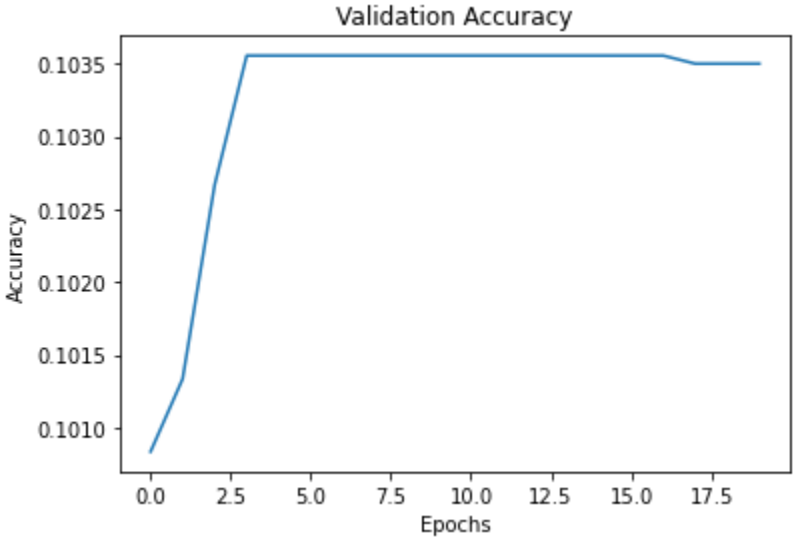
a) Gradient Descent with momentum:

Training loss sees a decrease after an initial jump whereas validation loss and accuracy archives a stable state after the initial jump, this could be explained because of the fact that the training improves after many epochs, but the uncertainty in training accuracy (increasing) suggests more number of epochs needed for better training.



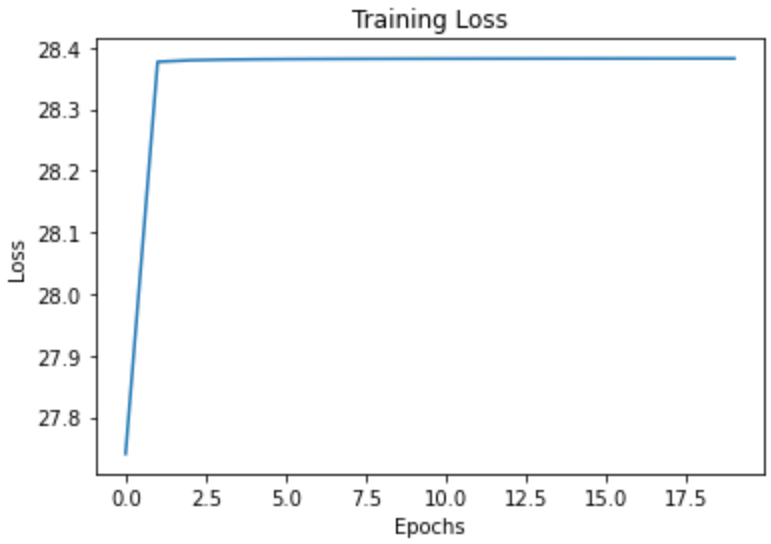


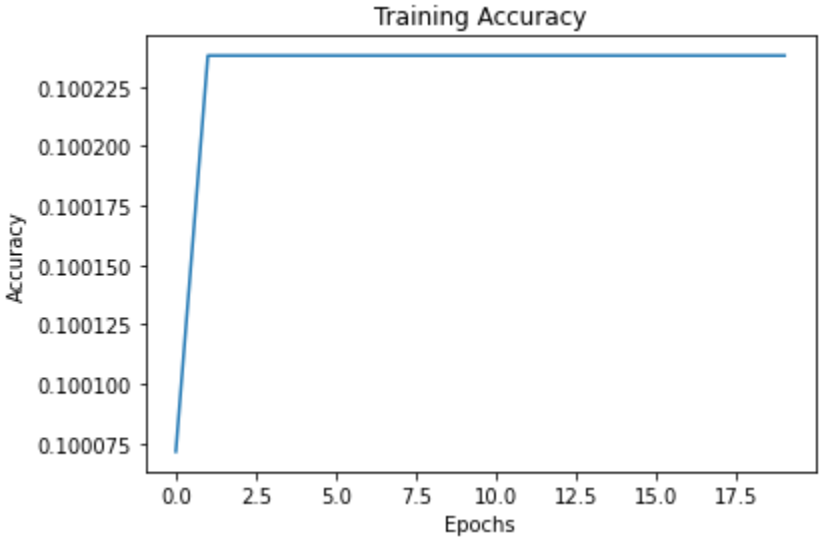


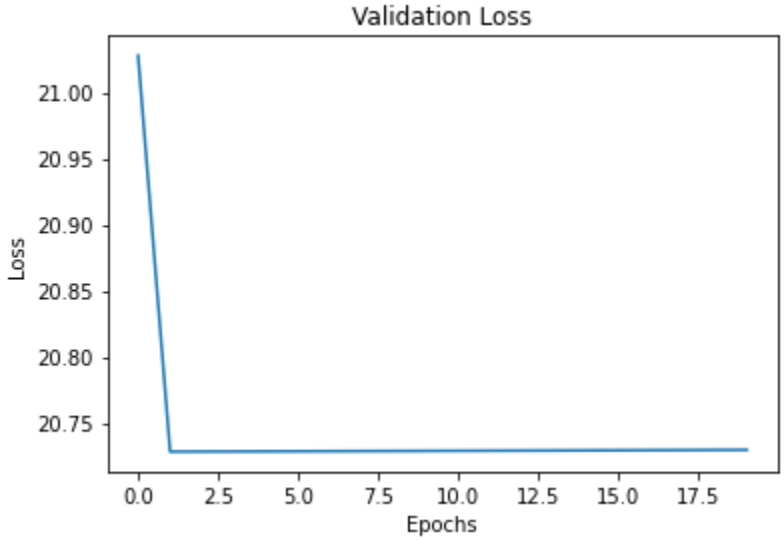


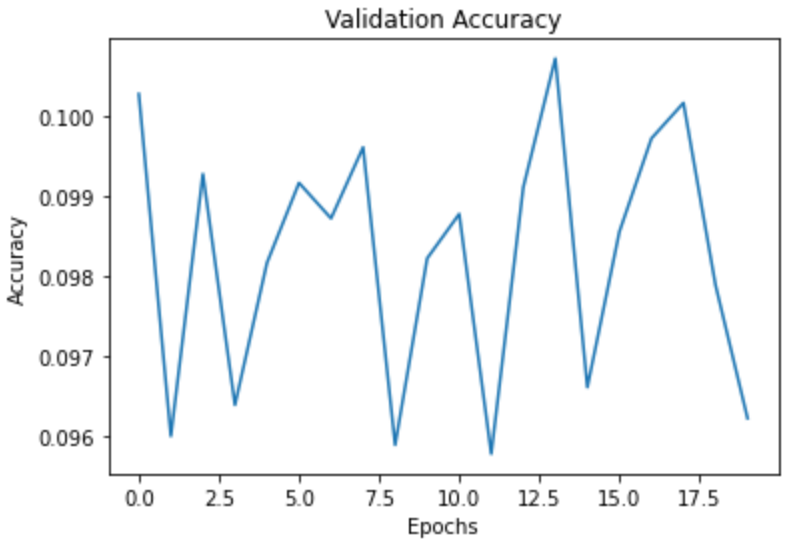
b) Nestrov's Accelerated Gradient

The training loss and accuracy, and validation loss gets stable after a certain number of epochs and has slightly better results than simple gradient descent with momentum but the uncertainty in validation accuracy suggests there should be an improvement in the choice of hyperparameters and ofcourse, the number of epochs.


Training Loss
28.4
28.3
28.2
28.1
28.0
27.9
27.8
Loss
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs

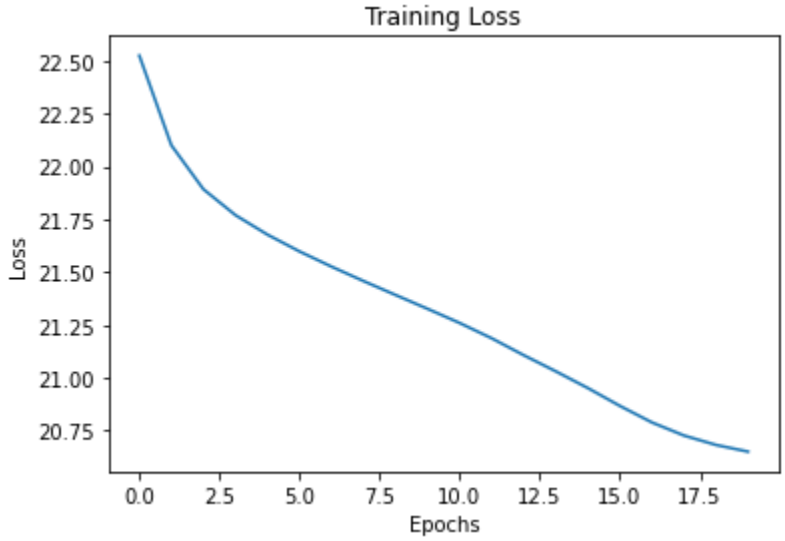

Training Accuracy
0.100225
0.100200
0.100175
0.100150
0.100125
0.100100
0.100075
Accuracy
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs

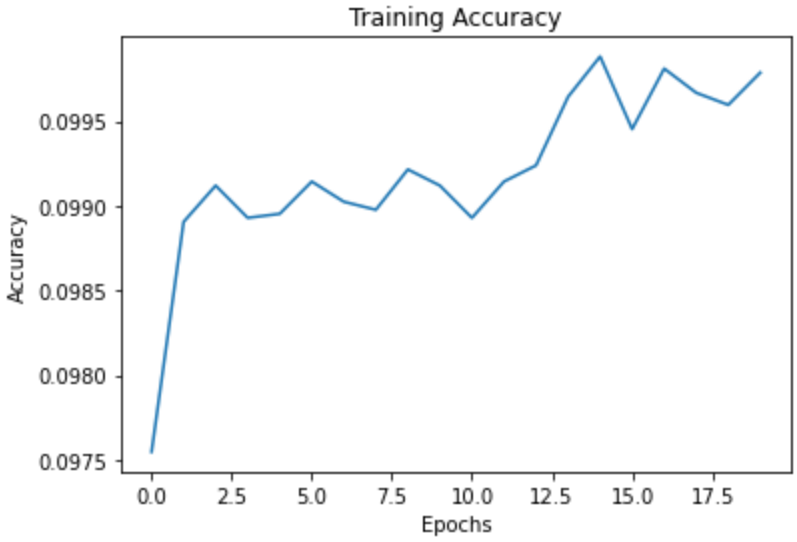

Validation Loss
21.00
20.95
20.90
20.85
20.80
20.75
Loss
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs

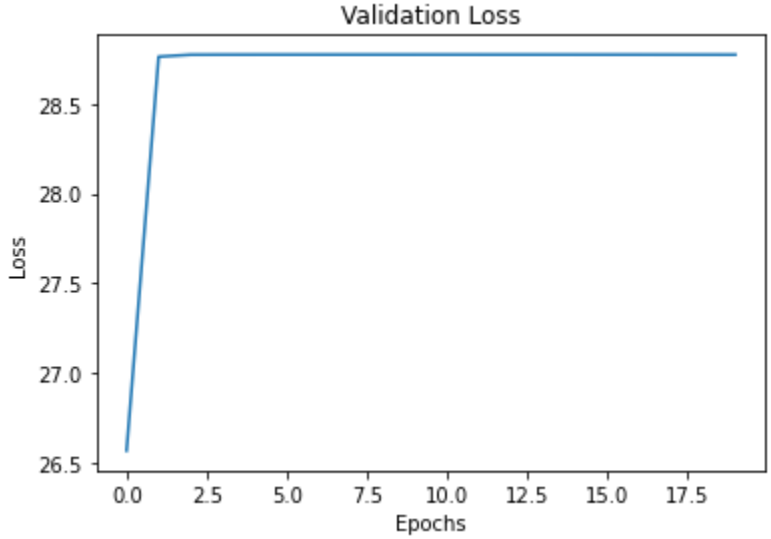


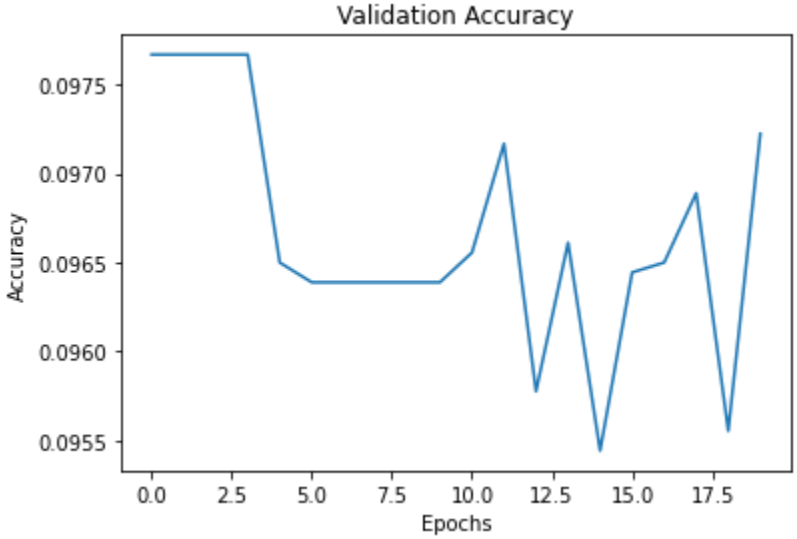
c) AdaGrad

The improvement in the learning rate makes the training loss decrease monotonically, training accuracy increases (not monotonically, but satisfactorily) due to similar reasons. The validation loss settles after a certain number of epochs but there seems to be an uncertainty in the validation accuracy, but that too gets to increase after some epochs; all this suggesting there should be a good combination of hyperparameters to be used. The results improve as compared to the Nestrov's accelerated gradient descent method.



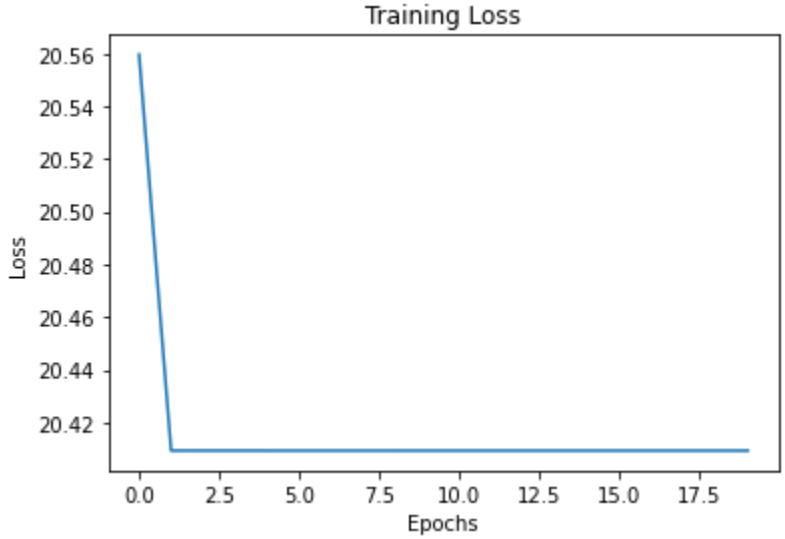

Training Accuracy
Accuracy
0.0995
0.0990
0.0985
0.0980
0.0975
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs

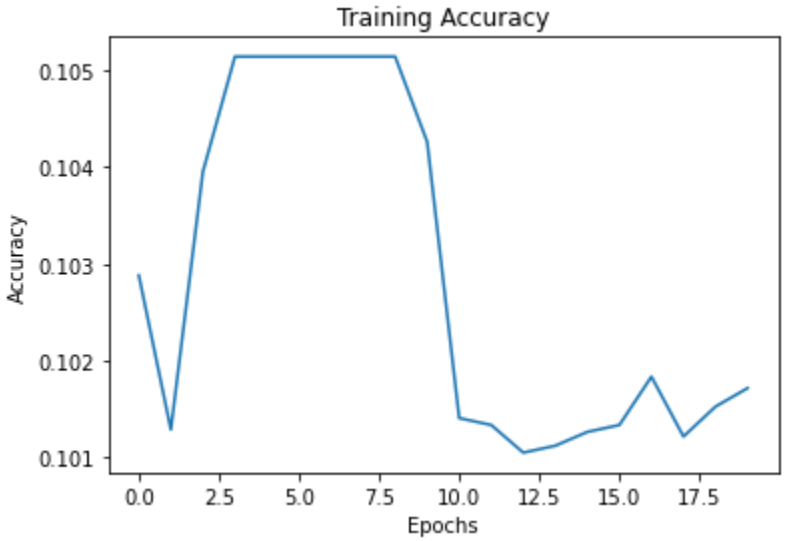

Validation Loss
Loss
28.5
28.0
27.5
27.0
26.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs

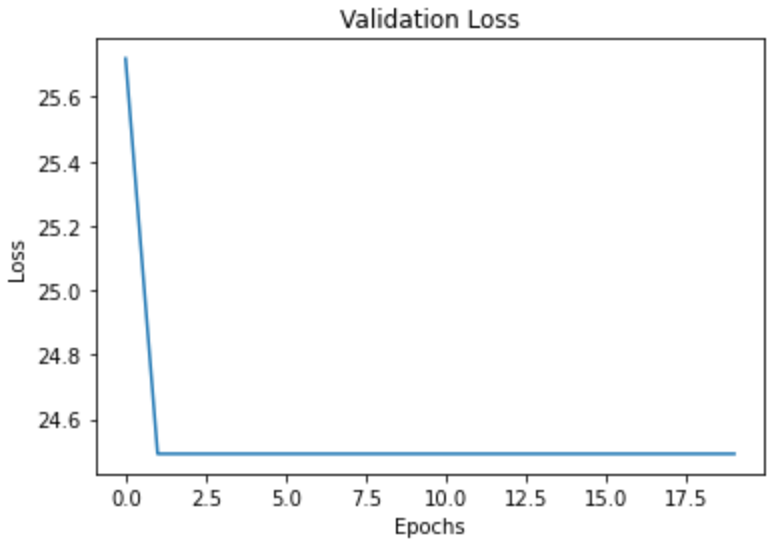

Validation Accuracy
Accuracy
0.0975
0.0970
0.0965
0.0960
0.0955
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs

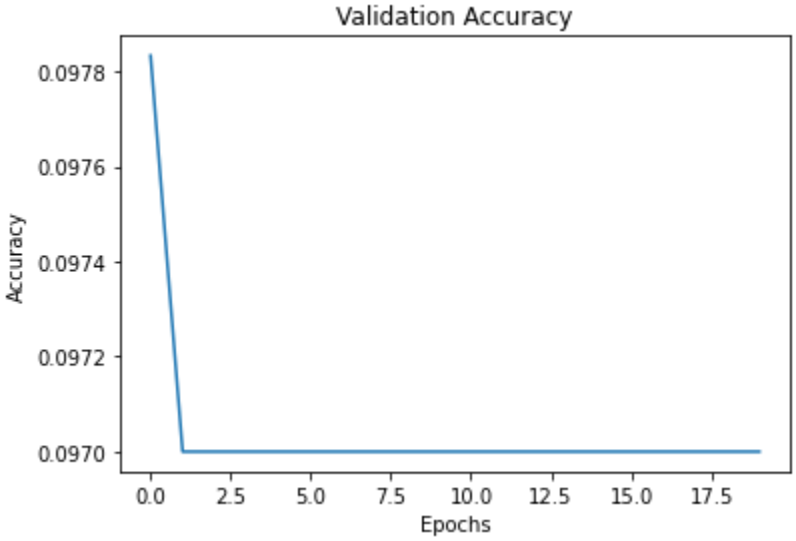
d) RMS Prop

The training loss gets to a higher point and then decreases to get stable after a number of epochs, but the training accuracy getting unstable after every few epochs suggests the problem that usually occurs in RMSProp technique, that is, the accumulation of squared sum of gradients, which ultimately reduces training, and it is due to this reason that the validation accuracy and loss decreases and gets stable at a lower point.



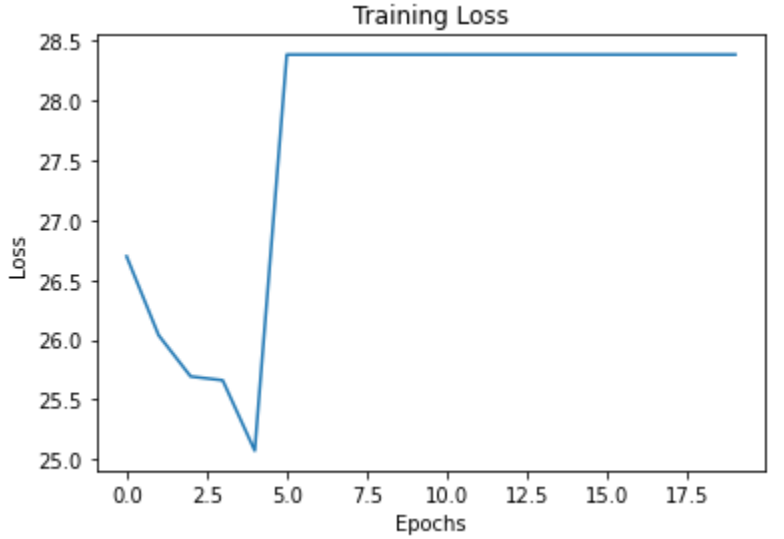


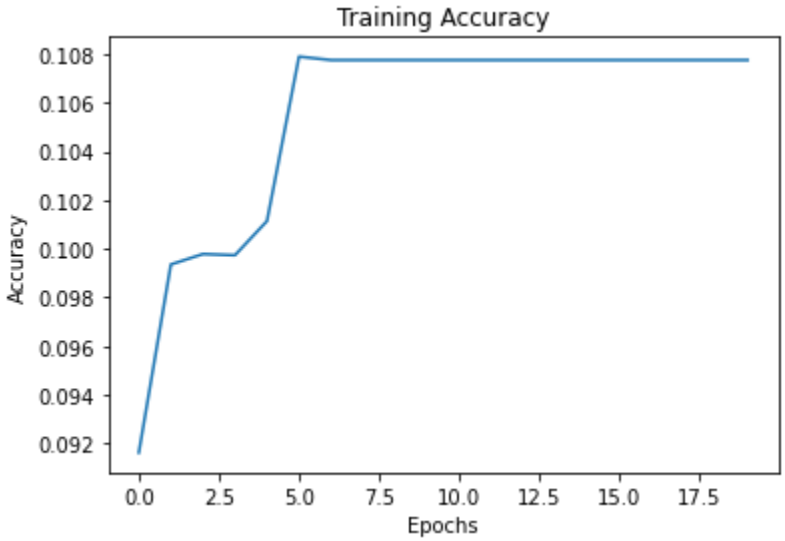

Validation Loss
25.6
25.4
25.2
25.0
24.8
24.6
Loss
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs

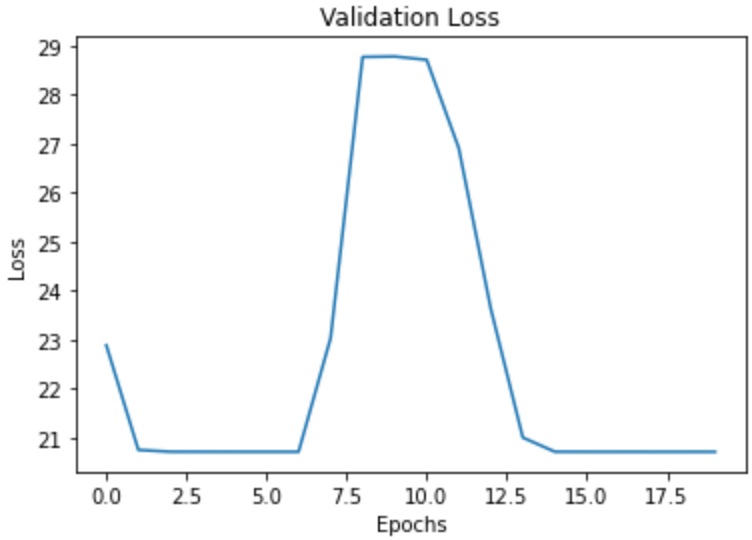

Validation Accuracy
0.0978
0.0976
0.0974
0.0972
0.0970
Accuracy
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs

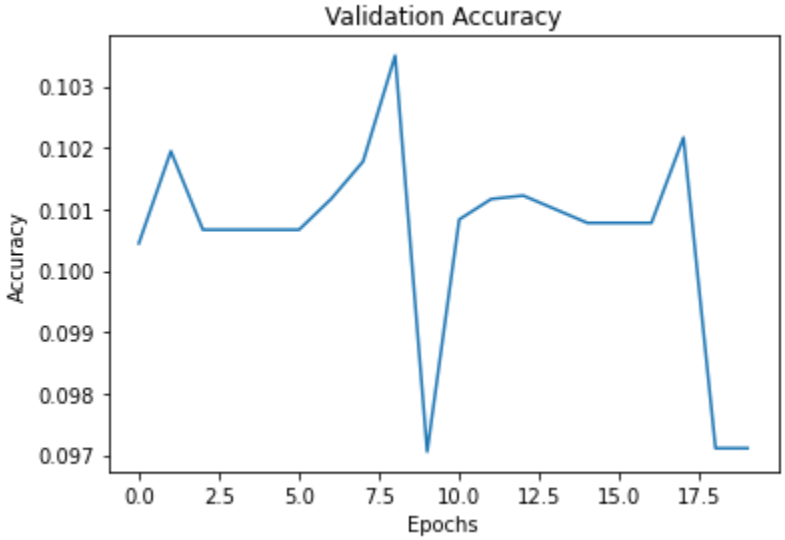
e) Adam

The training seems to increase but gets settled at a value, while the training accuracy increases and is higher than that seen in any of the previous methods, proving the fact that Adam in fact is the optimal choice as an optimizer. The validation loss curve achieves the minimum value of error ever seen in the above optimizers, and the validation accuracy being uncertain at many points again suggests an improvement in the hyperparameters to be done from our side.



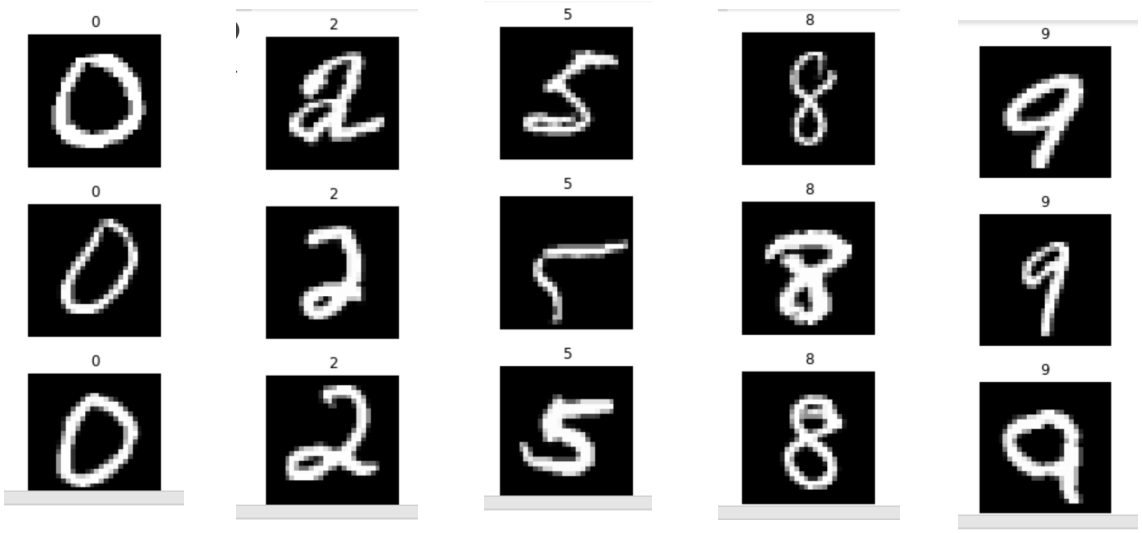



Validation Loss
Loss
Epochs
29
28
27
26
25
24
23
22
21
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Validation Accuracy
Accuracy
Epochs
0.103
0.102
0.101
0.100
0.099
0.098
0.097
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

2. Convolution Neural Network (CNN)

1. Visualization of 5 Random images from 5 different digits.



2. We created a CNN architecture having a filter of size 5x5 with 10 feature maps followed by another filter of size 2x2 with 20 feature maps Then added a max pooling to each convolution layer Then connected it to dense neural network which has a linear layer with 50 neurons and relu as activation function and a classification head layer with 10 neurons and softmax for the given problem of 10 classes(digit zero to nine) classification problem.

Optimizer = adam
loss = sparse categorical cross entropy

Epoch wise losses and accuracy for train set :

```
cnn1_train_history = cnn1.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
1500/1500 [==============================] - 29s 18ms/step - loss: 0.2382 - accuracy: 0.9291
Epoch 2/10
1500/1500 [==============================] - 30s 20ms/step - loss: 0.0762 - accuracy: 0.9754
Epoch 3/10
1500/1500 [==============================] - 27s 18ms/step - loss: 0.0571 - accuracy: 0.9823
Epoch 4/10
1500/1500 [==============================] - 27s 18ms/step - loss: 0.0446 - accuracy: 0.9860
Epoch 5/10
1500/1500 [==============================] - 28s 19ms/step - loss: 0.0366 - accuracy: 0.9879
Epoch 6/10
1500/1500 [==============================] - 27s 18ms/step - loss: 0.0297 - accuracy: 0.9906
Epoch 7/10
1500/1500 [==============================] - 27s 18ms/step - loss: 0.0258 - accuracy: 0.9919
Epoch 8/10
1500/1500 [==============================] - 27s 18ms/step - loss: 0.0224 - accuracy: 0.9925
Epoch 9/10
1500/1500 [==============================] - 27s 18ms/step - loss: 0.0182 - accuracy: 0.9939
Epoch 10/10
1500/1500 [==============================] - 27s 18ms/step - loss: 0.0164 - accuracy: 0.9943
```

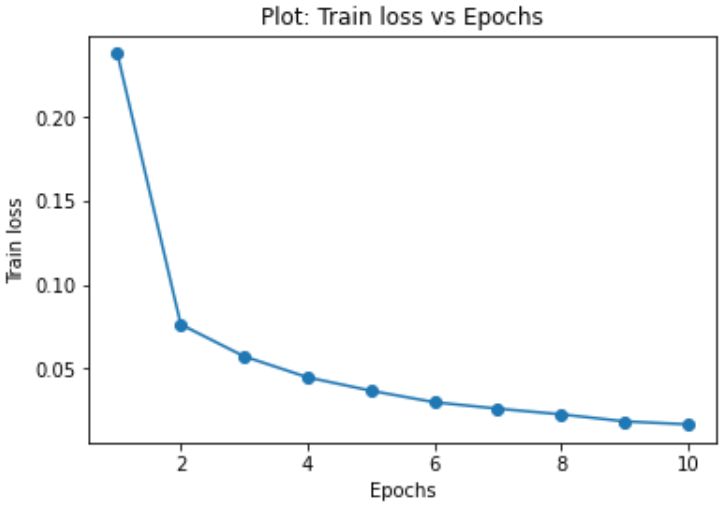
Epoch wise losses and accuracy for validation set :

```
cnn1_valid_history = cnn1.fit(X_valid, y_valid, epochs=10)
```

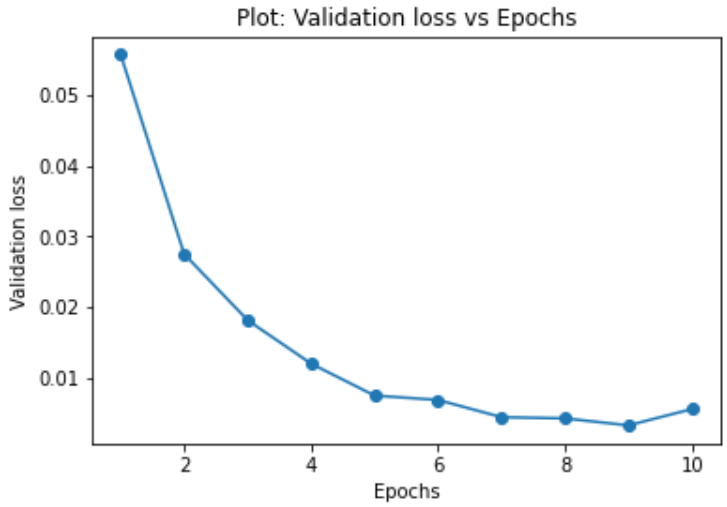
```
Epoch 1/10
375/375 [==============================] - 7s 20ms/step - loss: 0.0557 - accuracy: 0.9846
Epoch 2/10
375/375 [==============================] - 6s 17ms/step - loss: 0.0275 - accuracy: 0.9915
Epoch 3/10
375/375 [==============================] - 7s 20ms/step - loss: 0.0181 - accuracy: 0.9947
Epoch 4/10
375/375 [==============================] - 6s 16ms/step - loss: 0.0120 - accuracy: 0.9965
Epoch 5/10
375/375 [==============================] - 7s 20ms/step - loss: 0.0075 - accuracy: 0.9982
Epoch 6/10
375/375 [==============================] - 6s 16ms/step - loss: 0.0068 - accuracy: 0.9982
Epoch 7/10
375/375 [==============================] - 7s 20ms/step - loss: 0.0044 - accuracy: 0.9990
Epoch 8/10
375/375 [==============================] - 6s 17ms/step - loss: 0.0042 - accuracy: 0.9990
Epoch 9/10
375/375 [==============================] - 7s 19ms/step - loss: 0.0032 - accuracy: 0.9992
Epoch 10/10
375/375 [==============================] - 7s 18ms/step - loss: 0.0055 - accuracy: 0.9985
```

Plots:

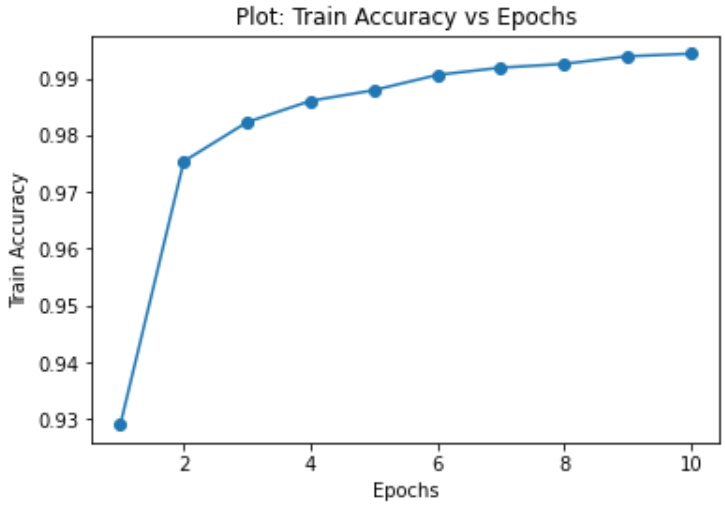
Plot : Train Loss vs Epochs



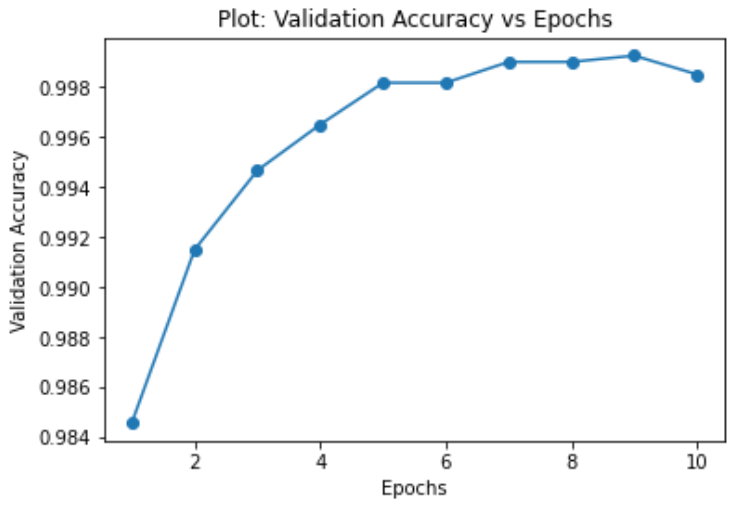
Plot : Validation Loss vs Epochs



Plot : Train Accuracy vs Epochs



Plot : Validation Accuracy vs Epochs



3. Data Augmentation

5-fold has been used as mentioned in the question for the 80:20 ratio. The loss gets minimized and the accuracy is maximized with each epoch for Both training and validation data We can see our code file. The accuracy and loss are around same
value for each of the 5 positional arguments. The Resize is used in the best model of part 2 and all other positional argumentation has almost the same accuracy for Mnist dataset as per model stats.