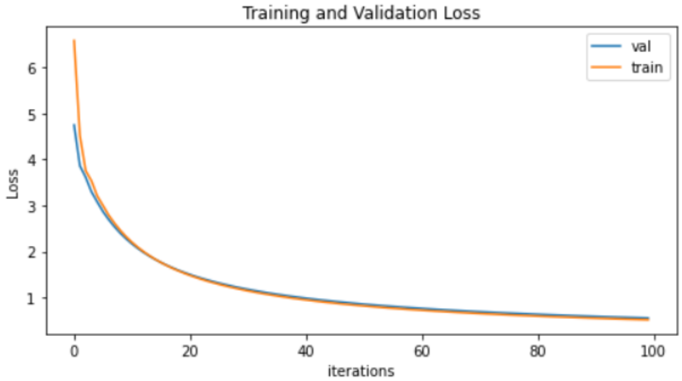
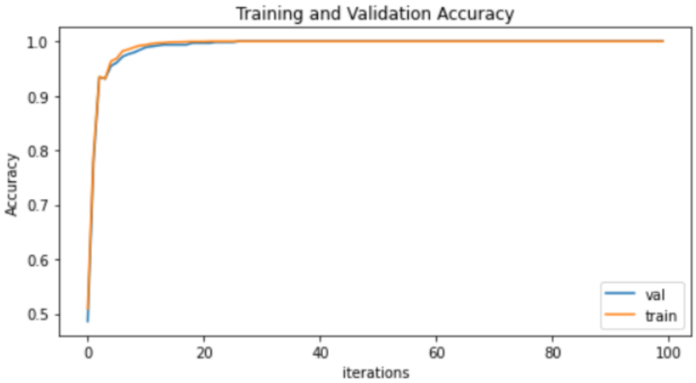
Assignment2_Task 1

Parameters: Epochs = 100, Learning Rate = 0.001

Results: Training Accuracy: 100%, Test Accuracy 100%





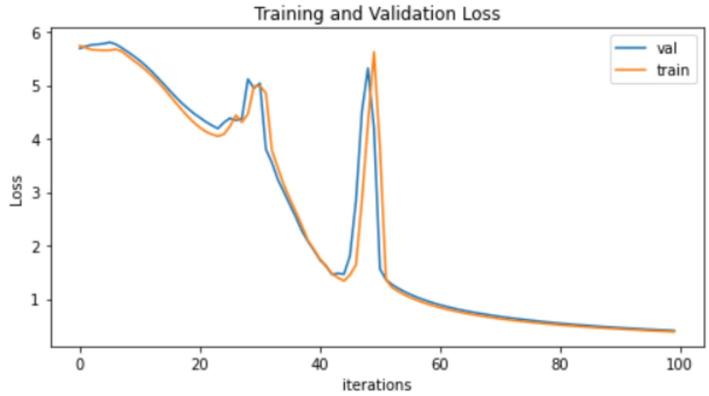
Parameters: Epochs = 100, Learning Rate = 0. 1

Results: Training Accuracy: 0, Test Accuracy 0

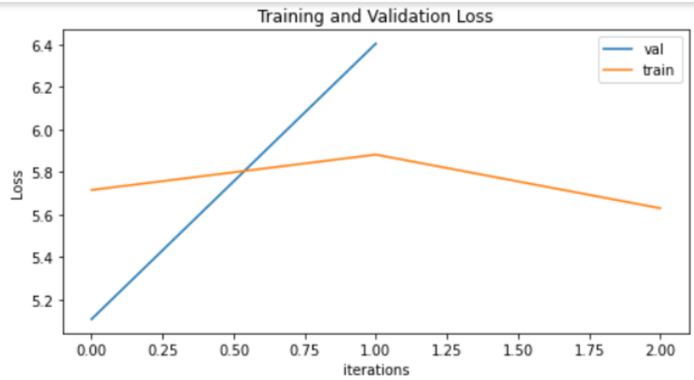
Analysis: As much as we increase the learning rate our model do not learn anything and not able to predict well.

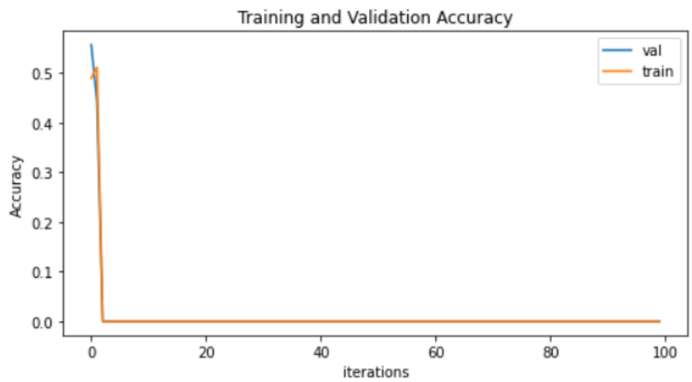
Assignment2_Task 2

Parameters: Epochs = 100, Learning Rate = 0. 005, Activation function =” Relu”
Results: Training Accuracy: 100%, Test Accuracy 100%

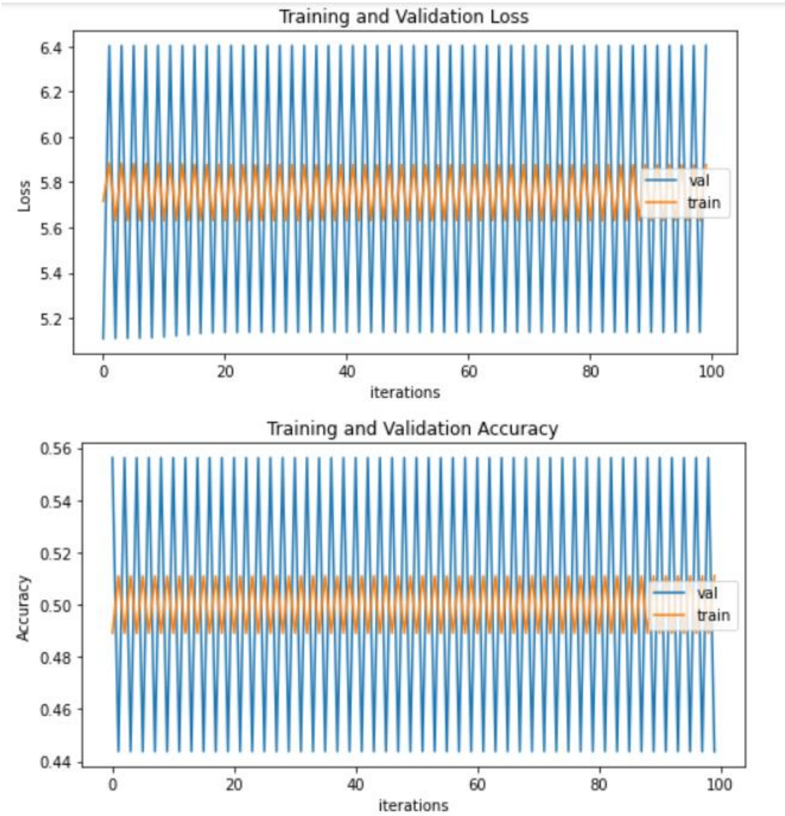


Parameters: Epochs = 100, Learning Rate = 0. 5, Function = “Relu”
Results: Training Accuracy: 0%, Test Accuracy: 0**%**



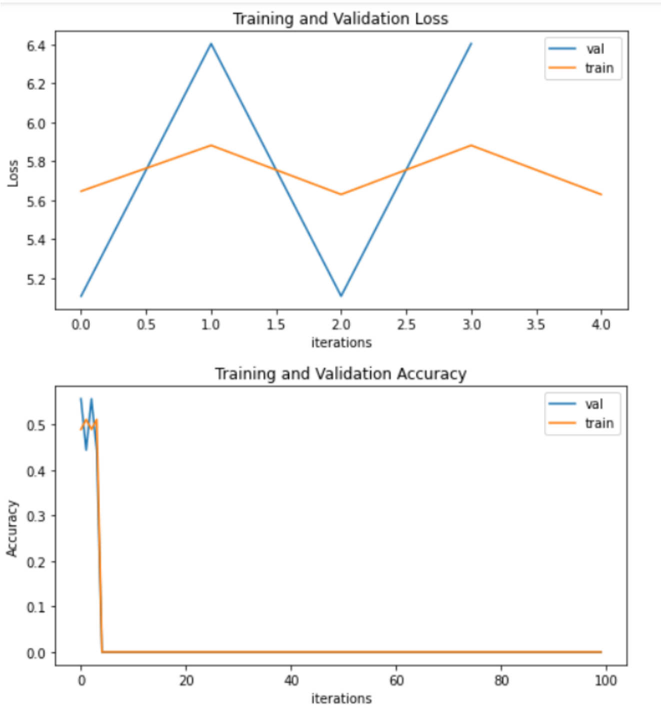


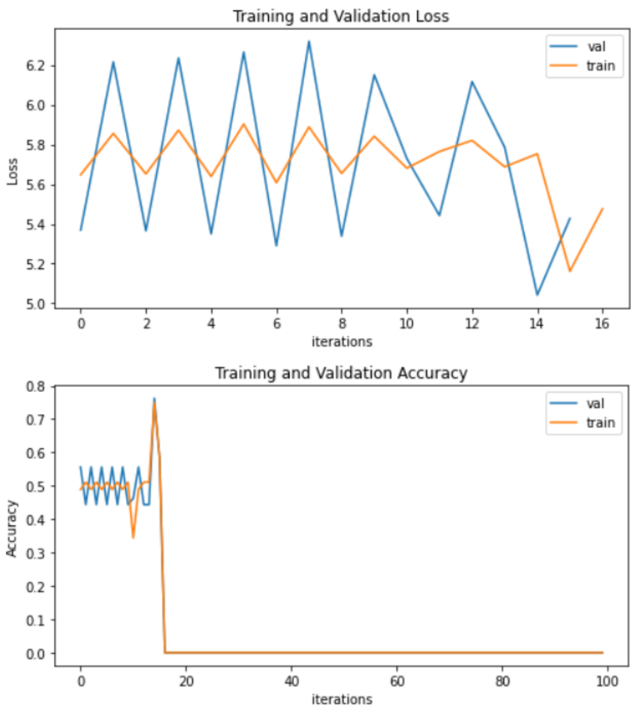
Parameters: Epochs = 100, Learning Rate = 0. 05, Function = "Relu"
Results: Training Accuracy: 0.49% - 0.51%, Test Accuracy: 0.51**%**



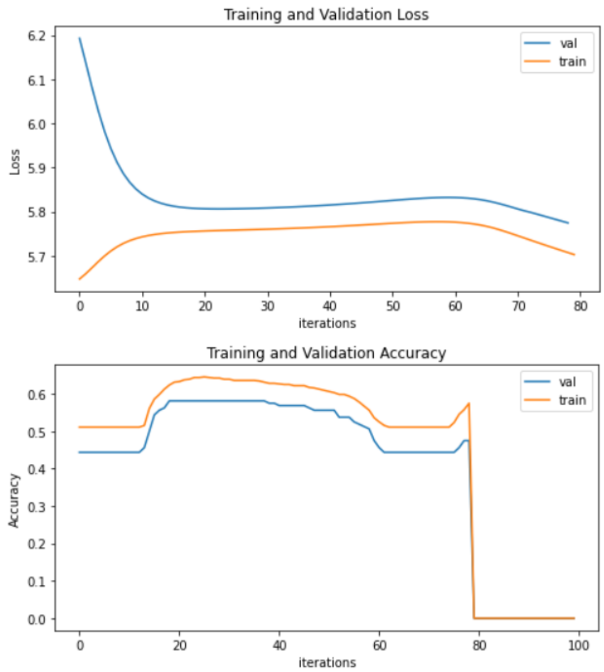
Analysis: When I increase the learning rate my model's learning is going to be decreased.

Parameters: Epochs = 100, Learning Rate = 0. 05 & 0.005, Function = “Sigmoid”
Results: Training Accuracy: 0%, Test Accuracy: 0%

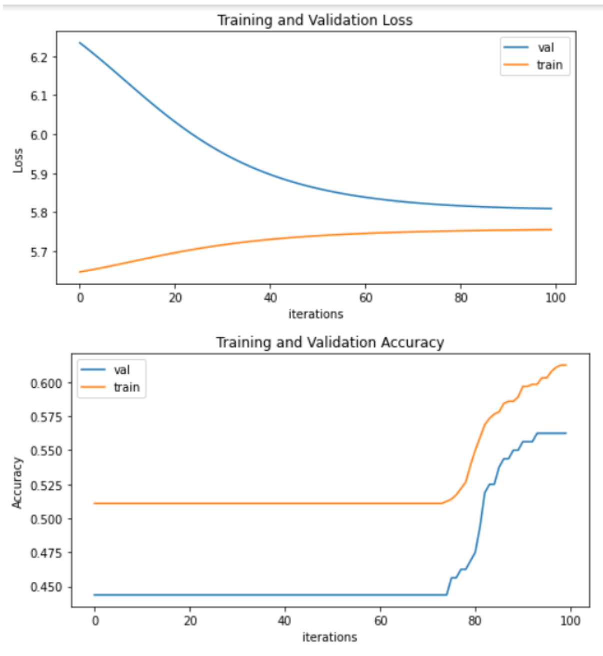




Parameters: Epochs = 100, Learning Rate = 0. 0005
Results: Training Accuracy: 0%, Test Accuracy: 0%

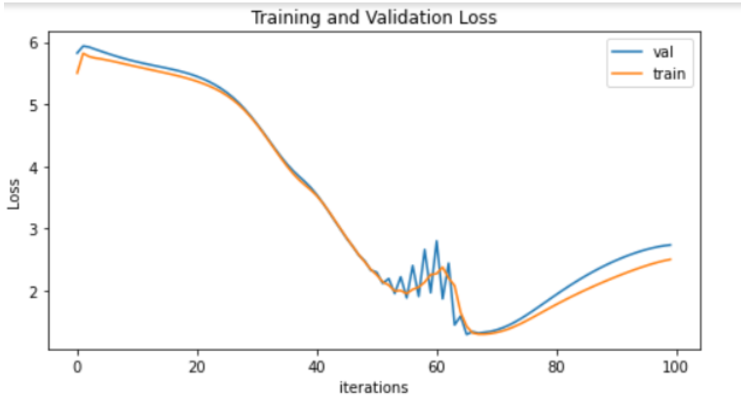


Learning Rate= 0.0001
Training Accuracy: 62%, Test Accuracy:61.5

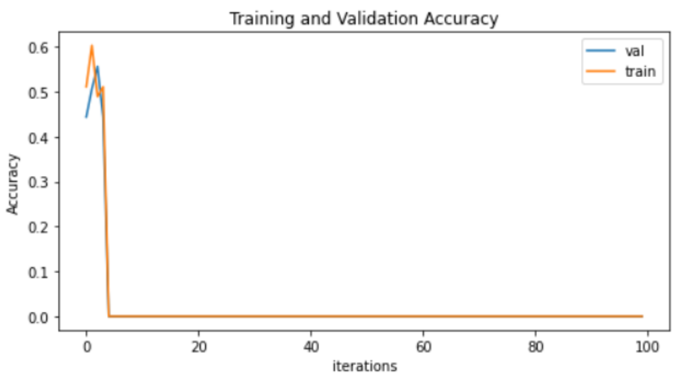


Analysis: My model does not perform well with sigmoid function. It is because of the vanishing gradient problem which is encountered when training artificial neural networks with gradient-based learning methods and backpropagation. In such methods, each of the neural network's weights receives an update proportional to the partial derivative of the error function with respect to the current weight in each iteration of training. The problem is that in some cases, the gradient will be vanishingly small, effectively preventing the weight from changing its value. In the worst case, this may completely stop the neural network from further training.

Parameters: Epochs = 100, Learning Rate = 0.005, Function = “tanh”
Results: Training Accuracy: 90%, Test Accuracy: 88%



Parameters: Epochs = 100, Learning Rate = 0.05, Function = “tanh”
Results: Training Accuracy: 0%, Test Accuracy: 0%

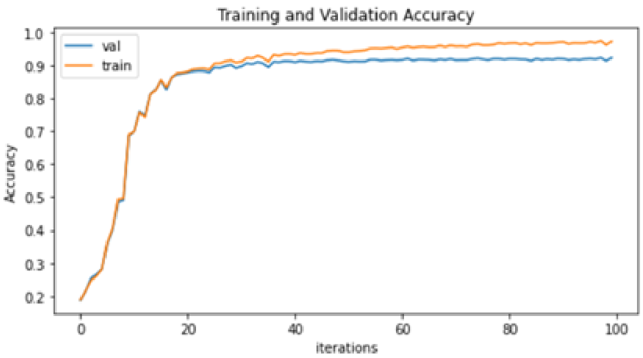


Assignment2_Task 3

Experiment with mean normalization (Best model)

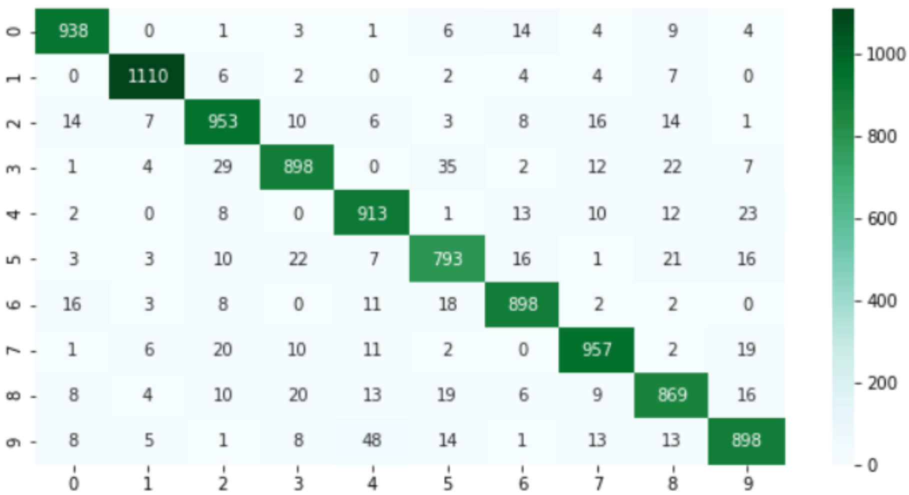
Parameters: Batch size = 20, learning Rate= 0.01, epoch= 100

Results: Training accuracy= 99%, validation accuracy= 92%



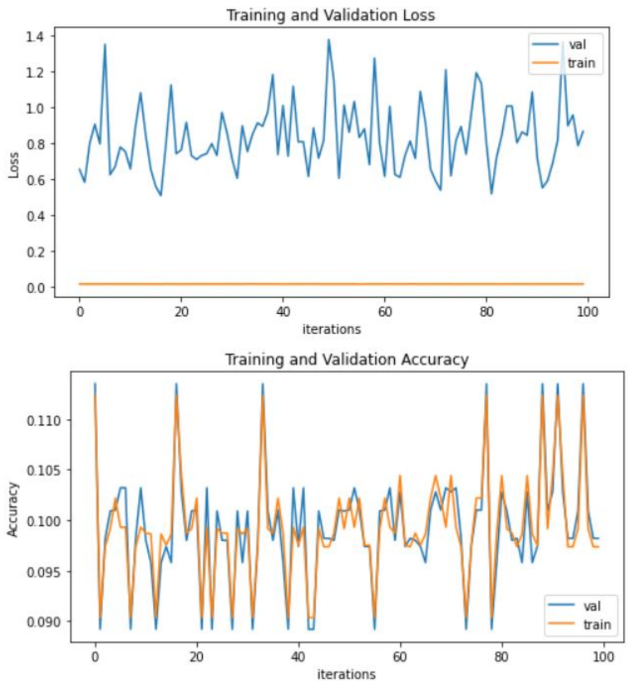

```
Test Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.96      0.95       980
           1       0.97      0.98      0.97      1135
           2       0.91      0.92      0.92      1032
           3       0.92      0.89      0.91      1010
           4       0.90      0.93      0.92       982
           5       0.89      0.89      0.89       892
           6       0.93      0.94      0.94       958
           7       0.93      0.93      0.93      1028
           8       0.89      0.89      0.89       974
           9       0.91      0.89      0.90      1009

    accuracy                           0.92     10000
   macro avg       0.92      0.92      0.92     10000
weighted avg       0.92      0.92      0.92     10000
```

Confusion Matrix:



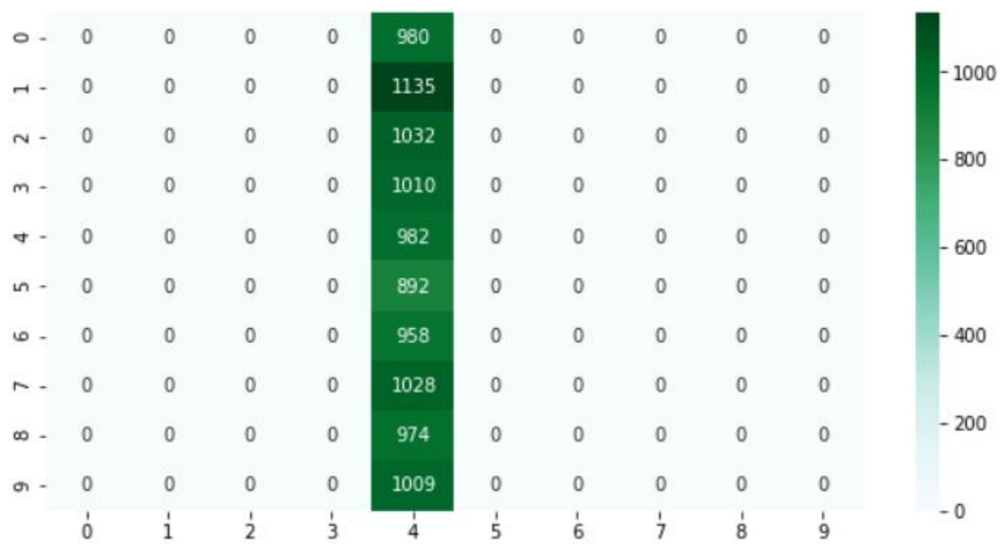
Experiment without mean normalization

Parameters: Batch size = 20, learning Rate= 0.01, epoch= 100
Results: Training accuracy= 10%, validation accuracy= 10%



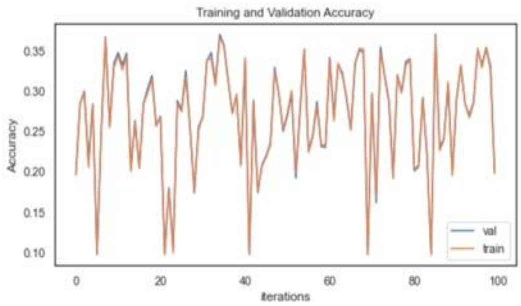
	precision	recall	f1-score	support
0	0.00	0.00	0.00	980
1	0.00	0.00	0.00	1135
2	0.00	0.00	0.00	1032
3	0.00	0.00	0.00	1010
4	0.10	1.00	0.18	982
5	0.00	0.00	0.00	892
6	0.00	0.00	0.00	958
7	0.00	0.00	0.00	1028
8	0.00	0.00	0.00	974
9	0.00	0.00	0.00	1009
accuracy			0.10	10000
macro avg	0.01	0.10	0.02	10000
weighted avg	0.01	0.10	0.02	10000

Confusion Matrix:

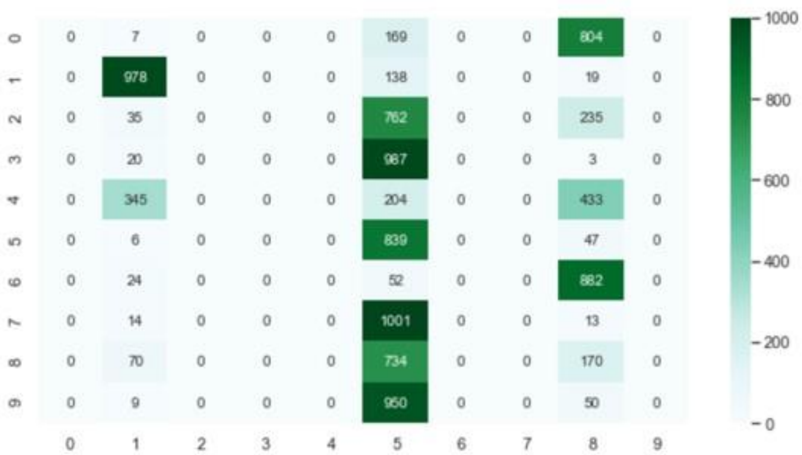


So if we did not use mean normalization our model will not learn anything.

Experiment with learning rate 1 with batch size 20

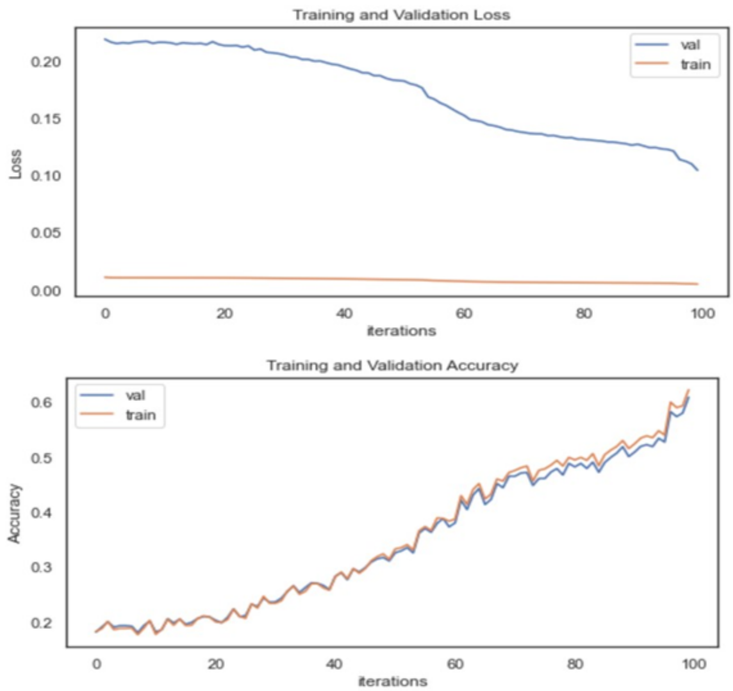


Confusion Matrix:

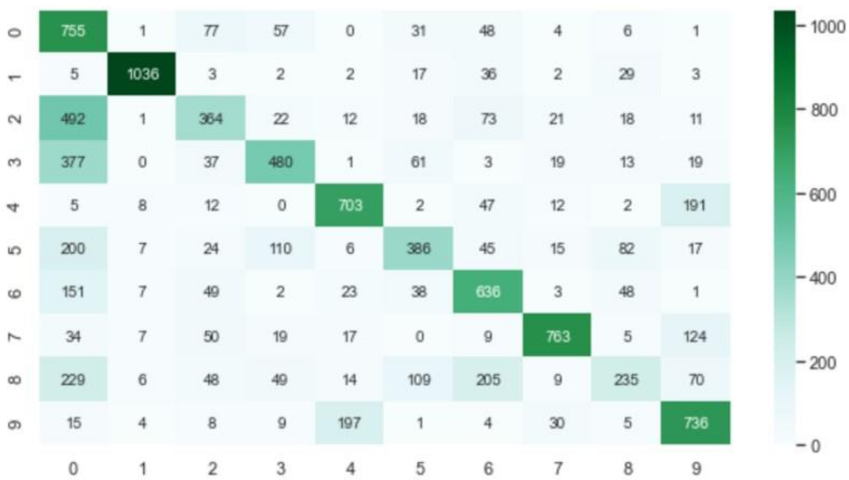


Training Accuracy = 35%
Test Accuracy = 20%
If we increase learning rate too high model will not learn properly

Experiment with learning Rate = 0.001, batch size = 20



<u>Confusion Matrix:</u>



Training Accuracy = 59%
Test Accuracy = 61%

If we have small learning rate to achieve best accuracy, we would have to increase the number of epochs.

Test accuracy of Best Model: 98%

T-distributed stochastic neighbor embedding (**T-SNE**):