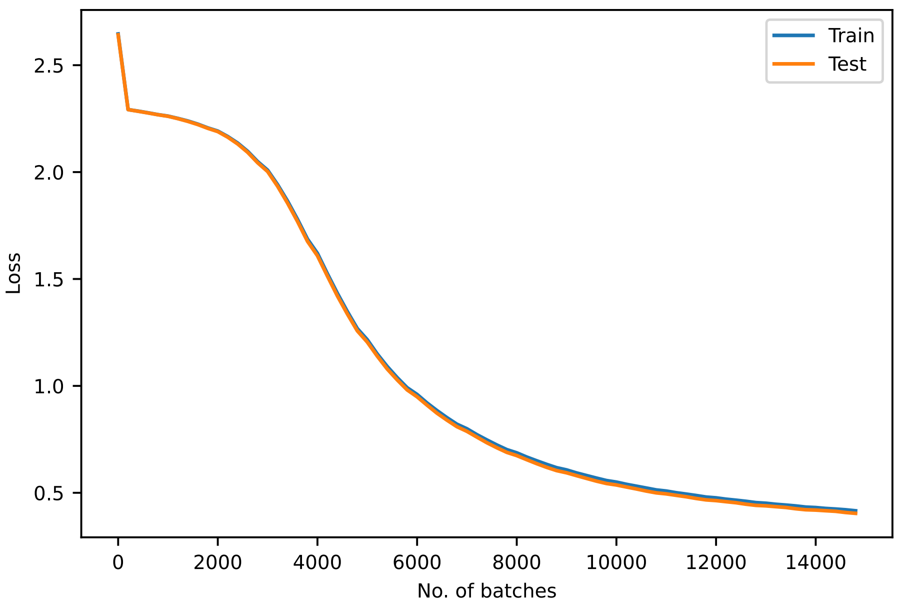
REPORT - DL A1

1 Backpropagation from scratch

1.1 Sigmoid Activation

```
layers = [784, 500, 250, 100, 10] # I/P, h1, h2, h3, O/P
batchSize = 64
epochs = 15
learningRate = 0.01
actFn = 'sigmoid'
```

1.1.1 Train and Test convergence



1.1.2 Confusion Matrix

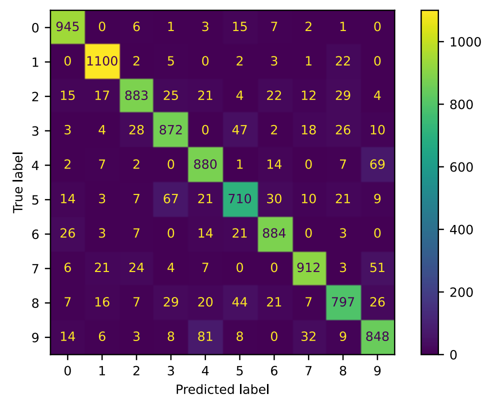


Figure 1.1: Test

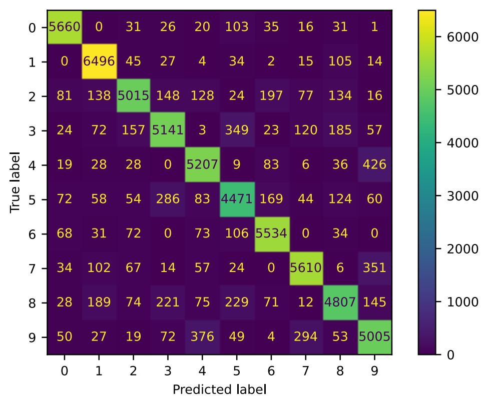


Figure 1.2: Train

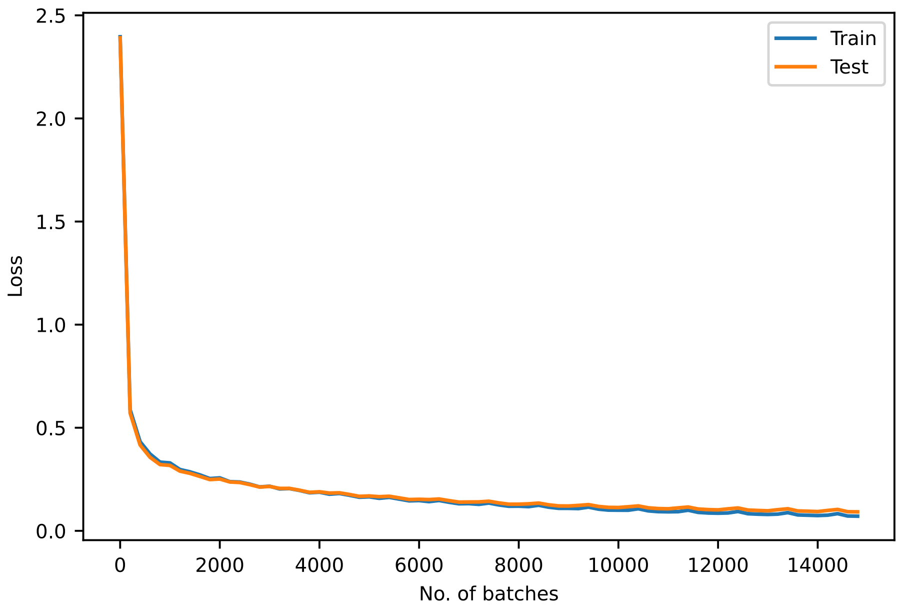
1.1.3 Net Accuracy

```
Test Accuracy: 88.31 %
Train Accuracy: 88.24 %
```

1.2 Tanh Activation

```
layers = [784, 500, 250, 100, 10] # I/P, h1, h2, h3, O/P
batchSize = 64
epochs = 15
learningRate = 0.01
actFn = 'Tanh'
```

1.2.1 Train and Test convergence



1.2.2 Confusion Matrix

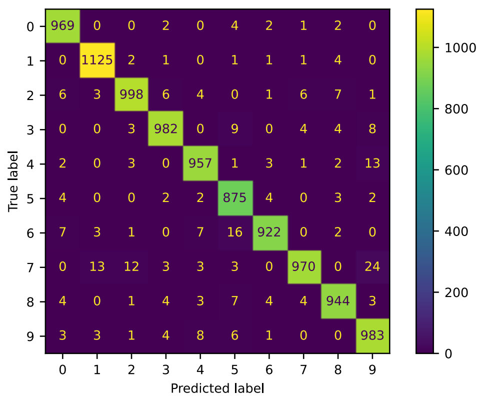


Figure 1.3: Test

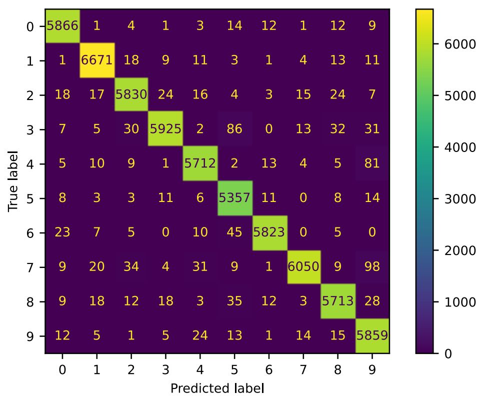


Figure 1.4: Train

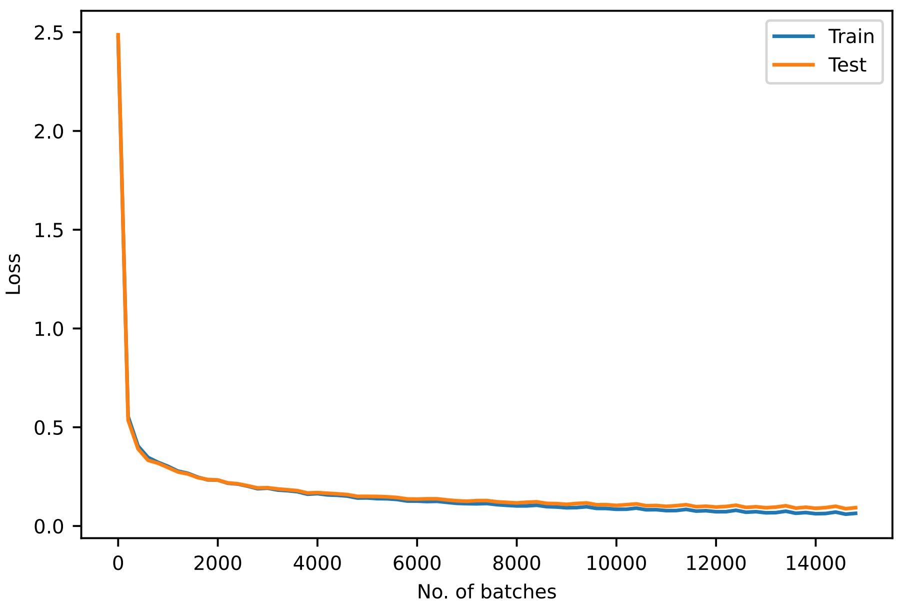
1.2.3 Net Accuracy

```
Test Accuracy: 97.25 %
Train Accuracy: 98.01 %
```

1.3 ReLU Activation

```
layers = [784, 500, 250, 100, 10] # I/P, h1, h2, h3, O/P
batchSize = 64
epochs = 15
learningRate = 0.01
actFn = 'ReLU'
```

1.3.1 Train and Test convergence



1.3.2 Confusion Matrix

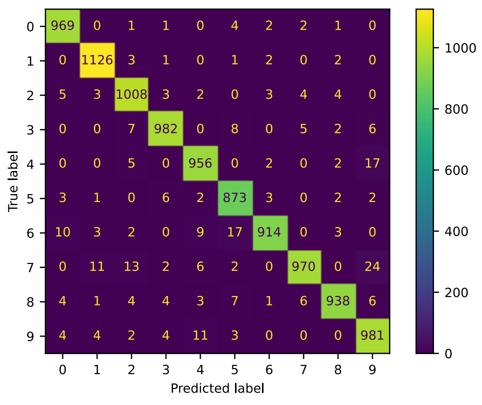


Figure 1.5: Test

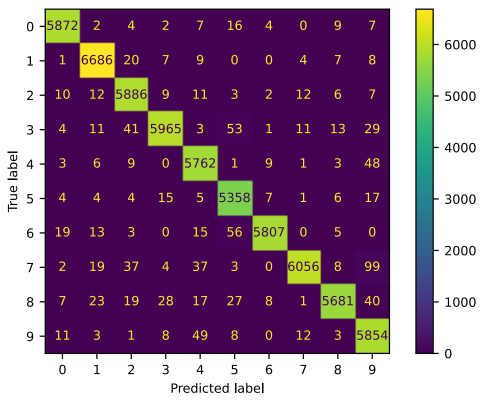


Figure 1.6: Train

1.3.3 Net Accuracy

```
Test Accuracy: 97.17 %
Train Accuracy: 98.21 %
```

1.4 Observations and comments

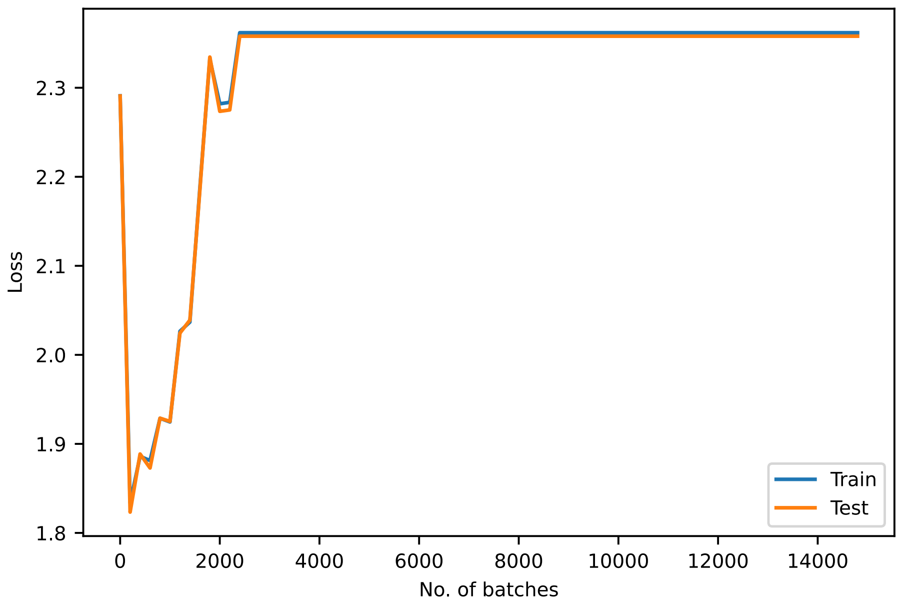
- Tanh and ReLU activation functions gave almost identical accuracies
- Tanh and ReLU converged quickly compared to sigmoid
- Sigmoid range is 0,1 and there is a vanishing gradient problem, so Sigmoid is theoretically the least accurate among the three. We observed the same in experimental results as well.
- Tanh has a better range -1,1 because of this reason it is theoretically superior to Sigmoid, we observe the same in our results.
- ReLU has an infinite range so it won't cause vanishing gradient problem, hence ReLU is theoretically the most superior activation function. Although in this case accuracy is almost similar to tanh. But ReLU has a lower runtime compared to Tanh
- The intial fall is extremely steep for Tanh and ReLU compared to sigmoid
- Some of the scenarios where the model is struggling is while classifying 7, 9 and 4, 9. This is the case with all the activations. This is happening because of the resemblence between the hand written characters

2 PyTorch

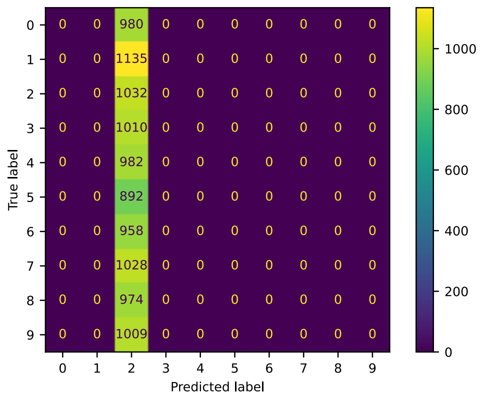
2.1 Same Hyperparameters with ReLU, ADAM

```
layers = [784, 500, 250, 100, 10] # I/P, h1, h2, h3, O/P
batchSize = 64
epochs = 15
learningRate = 0.01
actFn = 'ReLU'
```

2.1.1 Train and Test convergence

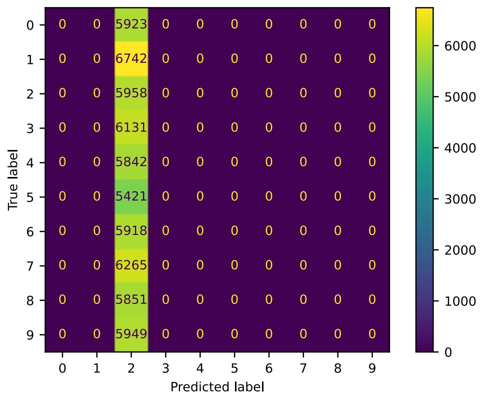


2.1.2 Confusion Matrix



	0	1	2	3	4	5	6	7	8	9
0	0	0	980	0	0	0	0	0	0	0
1	0	0	1135	0	0	0	0	0	0	0
2	0	0	1032	0	0	0	0	0	0	0
3	0	0	1010	0	0	0	0	0	0	0
4	0	0	982	0	0	0	0	0	0	0
5	0	0	892	0	0	0	0	0	0	0
6	0	0	958	0	0	0	0	0	0	0
7	0	0	1028	0	0	0	0	0	0	0
8	0	0	974	0	0	0	0	0	0	0
9	0	0	1009	0	0	0	0	0	0	0

Figure 2.1: Test



	0	1	2	3	4	5	6	7	8	9
0	0	0	5923	0	0	0	0	0	0	0
1	0	0	6742	0	0	0	0	0	0	0
2	0	0	5958	0	0	0	0	0	0	0
3	0	0	6131	0	0	0	0	0	0	0
4	0	0	5842	0	0	0	0	0	0	0
5	0	0	5421	0	0	0	0	0	0	0
6	0	0	5918	0	0	0	0	0	0	0
7	0	0	6265	0	0	0	0	0	0	0
8	0	0	5851	0	0	0	0	0	0	0
9	0	0	5949	0	0	0	0	0	0	0

Figure 2.2: Train

2.1.3 Net Accuracy

```
Test Accuracy: 10.32 %
Train Accuracy: 9.93 %
```

2.1.4 Comments

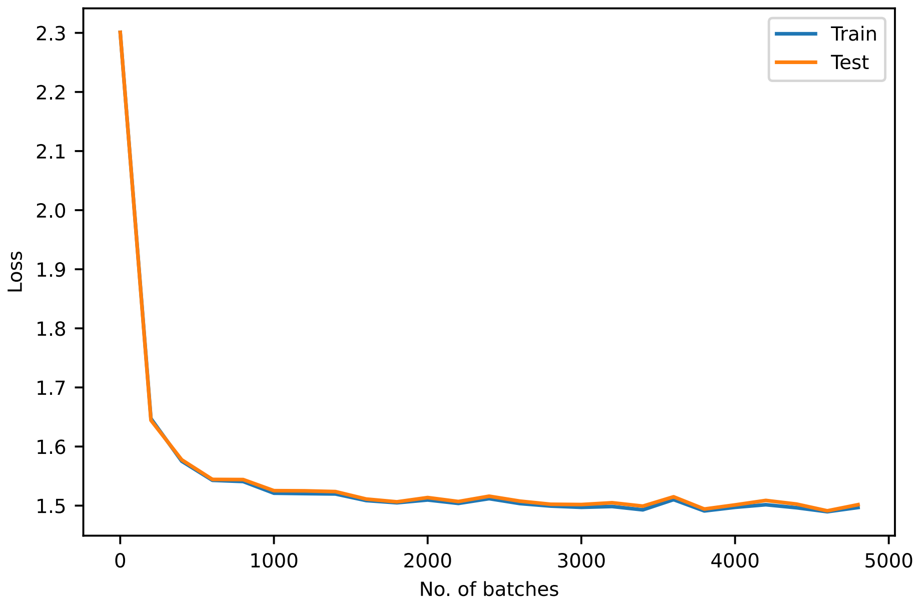
- In the Train and Test loss convergence we can see that loss first decreased then increased, it may be because the learning rate is high and we are overshooting across the minima
- We can avoid that by reducting the learning rate.

2.2 Tuned Hyperparameters with ReLU, ADAM

Learning Rate of 0.001 solved the overshooting problem, also numbers of epochs are reduced to 5, because the ADAM optimiser is converging quickly

```
layers = [784, 500, 250, 100, 10] # I/P, h1, h2, h3, O/P
batchSize = 64
epochs = 5
learningRate = 0.001
actFn = 'ReLU'
```

2.2.1 Train and Test convergence



2.2.2 Confusion Matrix

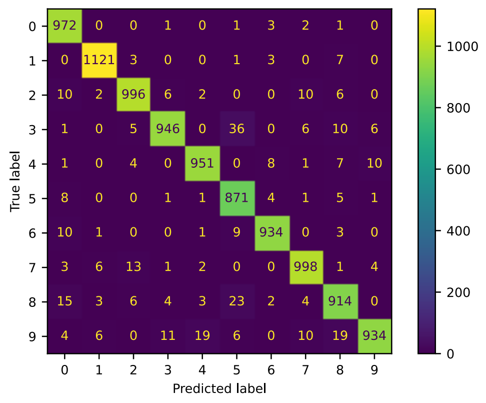


Figure 2.3: Test

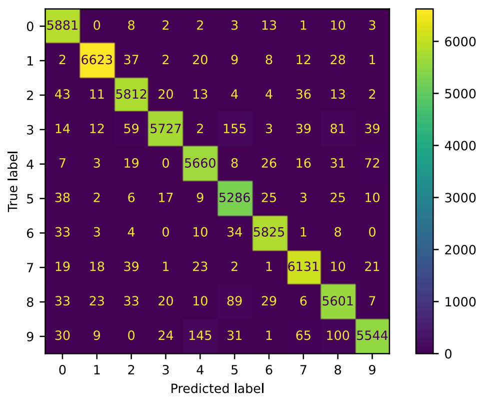


Figure 2.4: Train

2.2.3 Net Accuracy

```
Test Accuracy: 96.37 %
Train Accuracy: 96.82 %
```

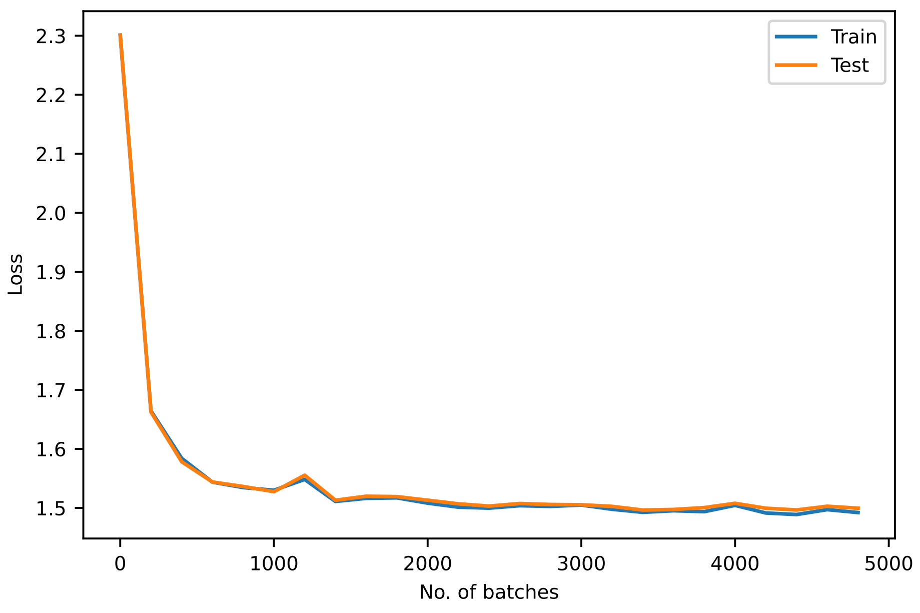
2.2.4 Comments:

- As guessed the problem of overshooting is resolved upon reducing the learning rate
- As we are using ADAM, it converged quickly compared to baseline model

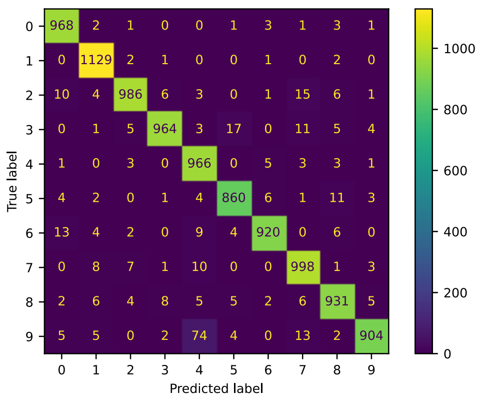
2.3 L2 Regularisation ReLU, ADAM

```
layers = [784, 500, 250, 100, 10] # I/P, h1, h2, h3, O/P
batchSize = 64
epochs = 5
learningRate = 0.001
weight decay = 1e-4
actFn = 'ReLU'
```

2.3.1 Train and Test convergence

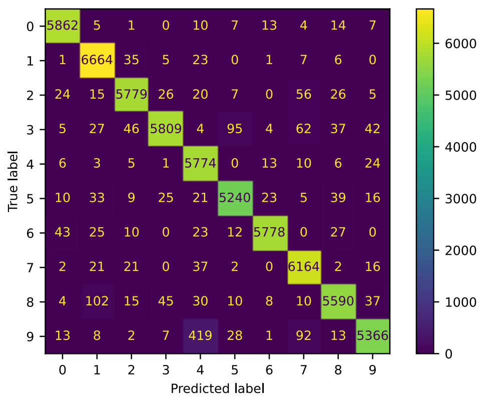


2.3.2 Confusion Matrix



	0	1	2	3	4	5	6	7	8	9
0	968	2	1	0	0	1	3	1	3	1
1	0	1129	2	1	0	0	1	0	2	0
2	10	4	986	6	3	0	1	15	6	1
3	0	1	5	964	3	17	0	11	5	4
4	1	0	3	0	966	0	5	3	3	1
5	4	2	0	1	4	860	6	1	11	3
6	13	4	2	0	9	4	920	0	6	0
7	0	8	7	1	10	0	0	998	1	3
8	2	6	4	8	5	5	2	6	931	5
9	5	5	0	2	74	4	0	13	2	904

Figure 2.5: Test



	0	1	2	3	4	5	6	7	8	9
0	5862	5	1	0	10	7	13	4	14	7
1	1	6664	35	5	23	0	1	7	6	0
2	24	15	5779	26	20	7	0	56	26	5
3	5	27	46	5809	4	95	4	62	37	42
4	6	3	5	1	5774	0	13	10	6	24
5	10	33	9	25	21	5240	23	5	39	16
6	43	25	10	0	23	12	5778	0	27	0
7	2	21	21	0	37	2	0	6164	2	16
8	4	102	15	45	30	10	8	10	5590	37
9	13	8	2	7	419	28	1	92	13	5366

Figure 2.6: Train

2.3.3 Net Accuracy

```
Test Accuracy: 96.26 %
Train Accuracy: 96.71 %
```

2.3.4 Comments:

- Here there is no overfitting to training data, hence the L2 regularisation didn't give much improvements.