Handwritten Digits Classification

CSE574 Introduction to Machine Learning
Spring 2019

Programming Assignment 2

Group No : 33
Manasi Kulkarni (Student No : 50288702)
Priyanka Pai (Student No : 50295903)
Kundan Kumar (Student No : 50301604)

Abstract :

This project involves getting familiar with neural networks and implementation of a multilayer perceptron neural network. It also involves comparison of performances of the same neural network against a deep neural network and a convolutional neural network using TensorFlow.

1. Introduction

Neural networks consist of a set of connected units. Each unit receives an input, processes it and generates an output, which in turn is provided to another unit. We are making use of single hidden layer. So our architecture has one input layer, one hidden layer and one output layer. We are using sigmoid activation function for each layer.

1.1 Concepts and definitions

Feed Forward

Feed forward models map an input to a category, and find out the best parameters for approximation. The flow of data is from input layers to the intermediate layers to the output layer.

Back Propagation

In back propagation, errors are propagated backwards from the output layer to the input layer. It makes use of gradient descent and updates weights at each step. For multilayer perceptron neural networks, it makes use of chain rule to compute the gradient.

Regularization

Regularization helps our neural network better at generalization. It helps in improving the performance of the model. It also helps to avoid overfitting of data.

Deep Neural Network

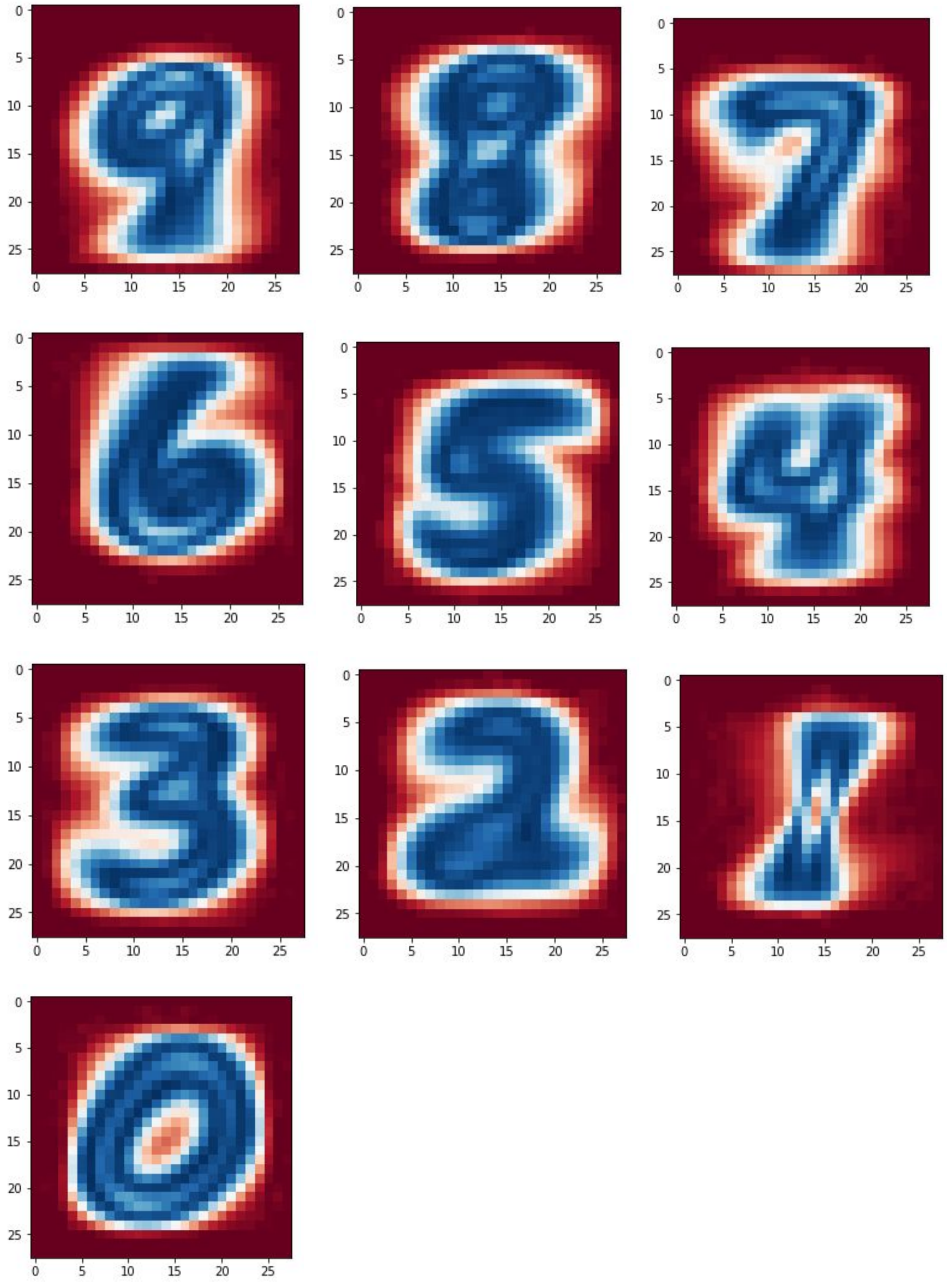
A Deep Neural Network is a neural network which has multiple layers between input and output layers.

Convolutional Neural Network

Convolutional Neural Network is a class of Deep Neural Network. Their applications are mostly in the image classification field. CNN consists of multiple hidden layers, which have an activation function, pooling layers etc.

2. Implementation

DataSet - Mnist Digits Data Set. This dataset has 60000 training data and another 10000 testing data. We divided the training data into two set 5:1 ratio for training and validation. We classify the dataset into 10 classes (one for each digit).
Following is the standard deviation image of each class.



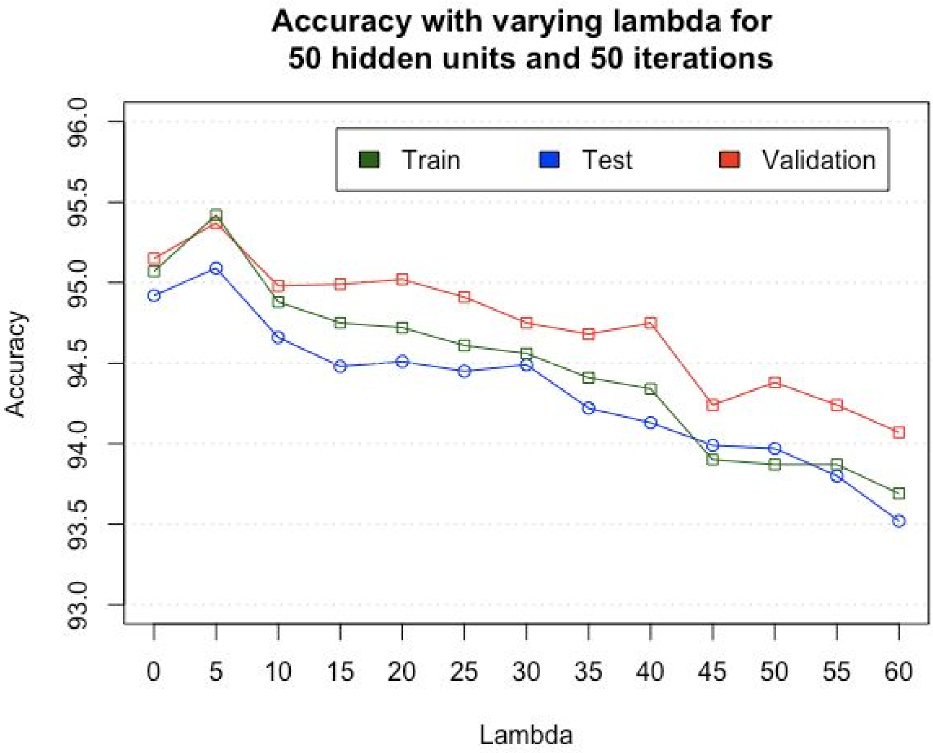
2.1. Choosing the hyper-parameter for Neural Network

For choosing the best hyper-parameter, we tested our implementation of neural network for different cases like varying the number of hidden units, varying the regularization factor(lambda) and testing our network for different number of iterations. Following are the results we observed in different cases :

1. Varying the value of lambda

Initially, we tested our network by varying lambda from 0 to 60 in intervals of 5 for 50 hidden units and 50 iterations.

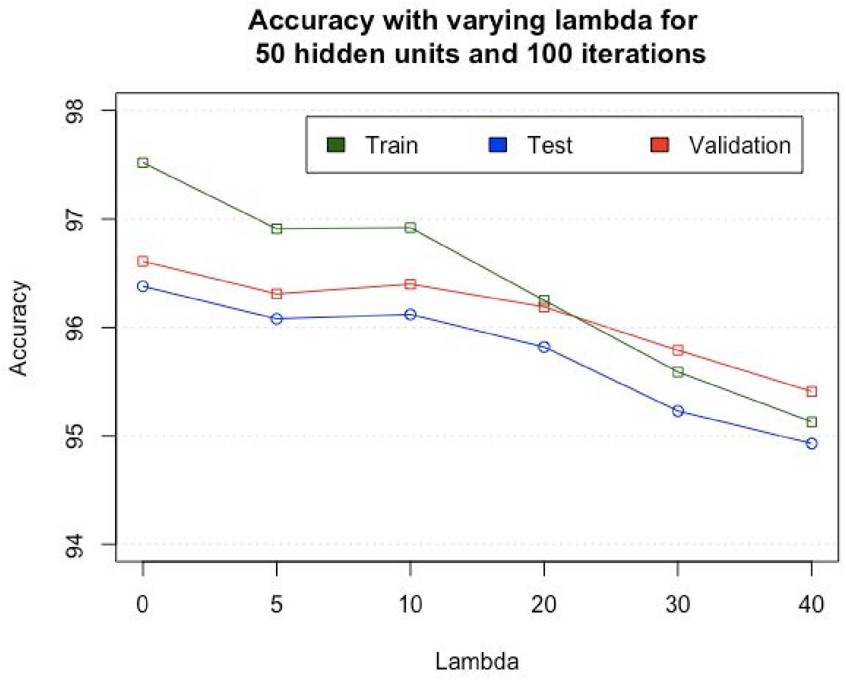
The results on train, test and validation data are :



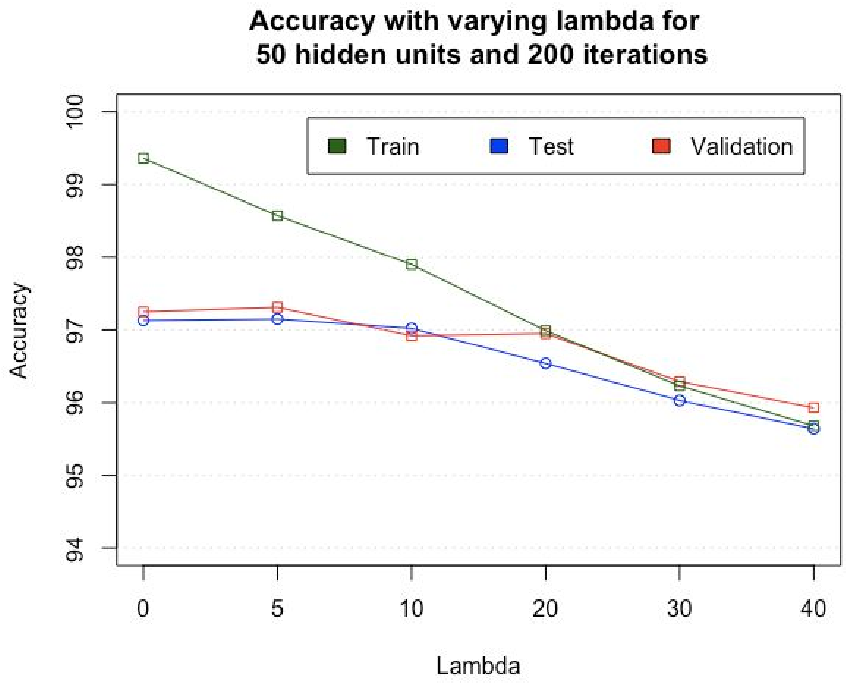
We observed that we get the maximum accuracy on test data and validation data when **lambda is 5**. The accuracy on test data with lambda as 5 is **95.09%.**

Lambda	**Accuracy**		
	Train data	**Validation data**	**Test data**
0	95.07	95.15	94.92
5	**95.42**	**95.37**	**95.09**
10	94.88	94.98	94.66
15	94.75	94.99	94.48
20	94.72	95.02	94.51
25	94.61	94.91	94.45
30	94.56	94.75	94.49
35	94.41	94.68	94.22
40	94.34	94.75	94.13
45	93.90	94.24	93.99
50	93.87	94.38	93.97
55	93.87	94.24	93.8
60	93.69	94.07	93.52

Later, to confirm our observations, we did the same by increasing the number of iterations. The results on train, test and validation data when lambda is in range (0,5,10,20,30,40) for **100 iterations** :



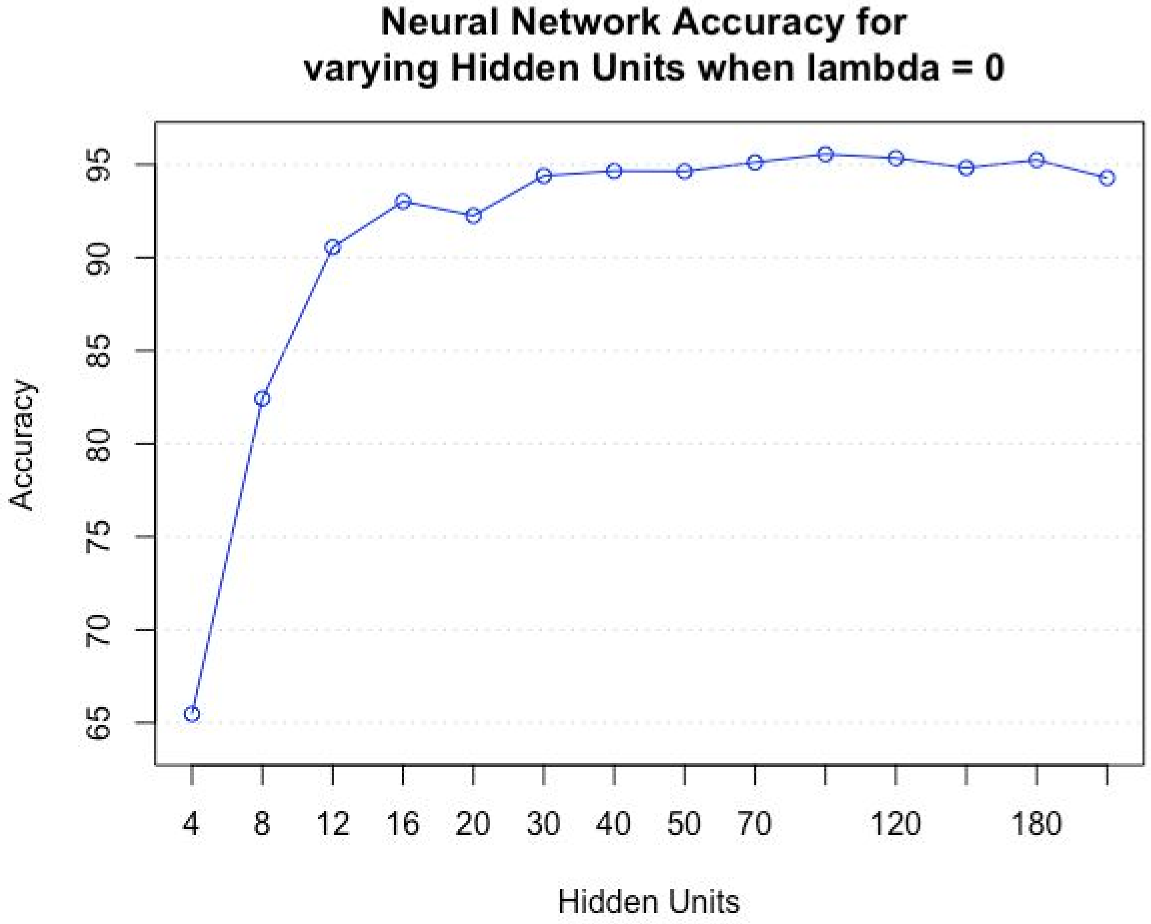
The results on train, test and validation data when lambda is in range (0,5,10,20,30,40) for **200 iterations** :



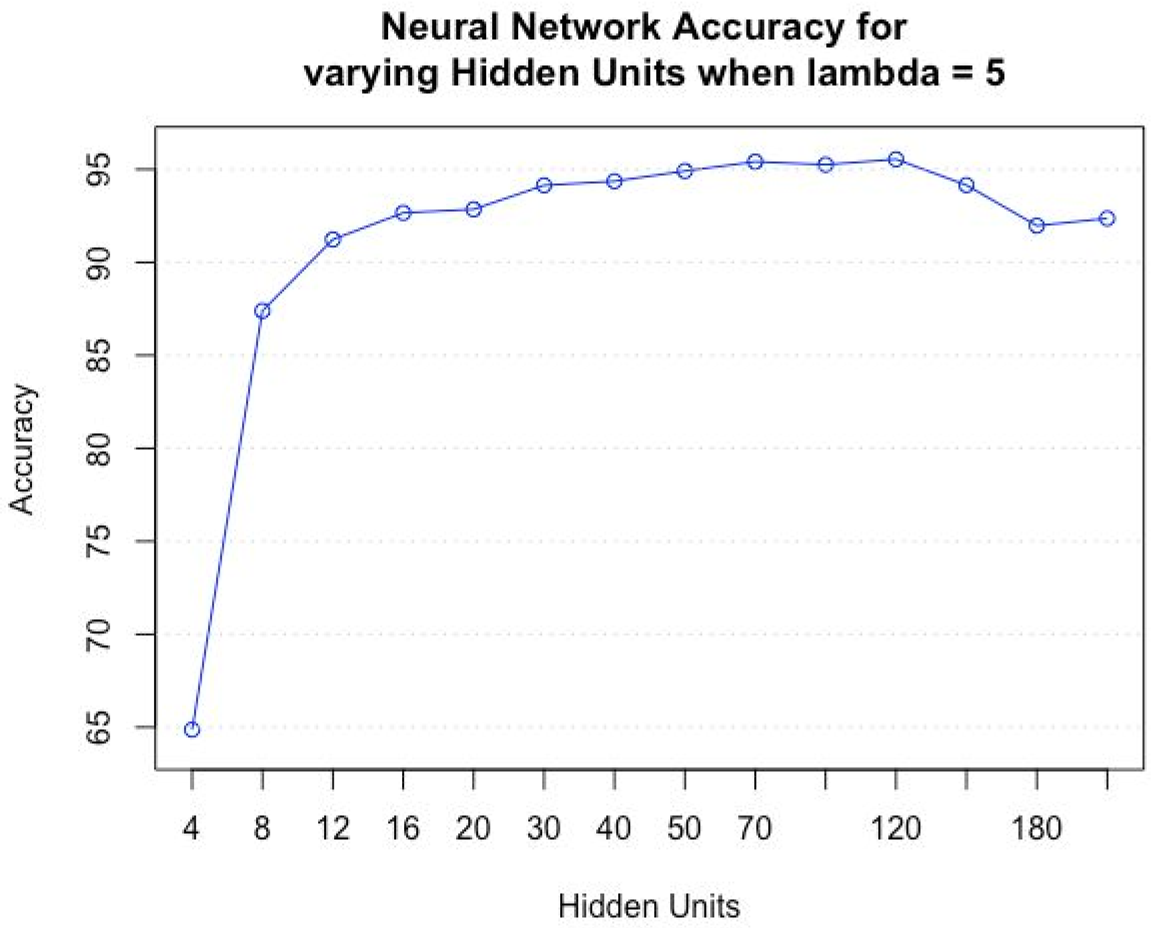
It can be observed that the **accuracy decreases with increasing lambda.**
Thus, our observation of lambda = 5 being optimal is confirmed by the above tests. For the further analysis of our network, we will be using 2 values of lambda : 0(no regularization) and 5(optimal).

2. Varying the number of hidden units for optimal lambda obtained from step 1
Using the optimal value of lambda we obtained from the previous step, we varied the number of hidden units as : 4,8,12,16,20,30,40,50,70,100,120,150,180 and 200

<u>The results on test data without regularization are :</u>



The results on test data with regularization(optimal lambda) are :



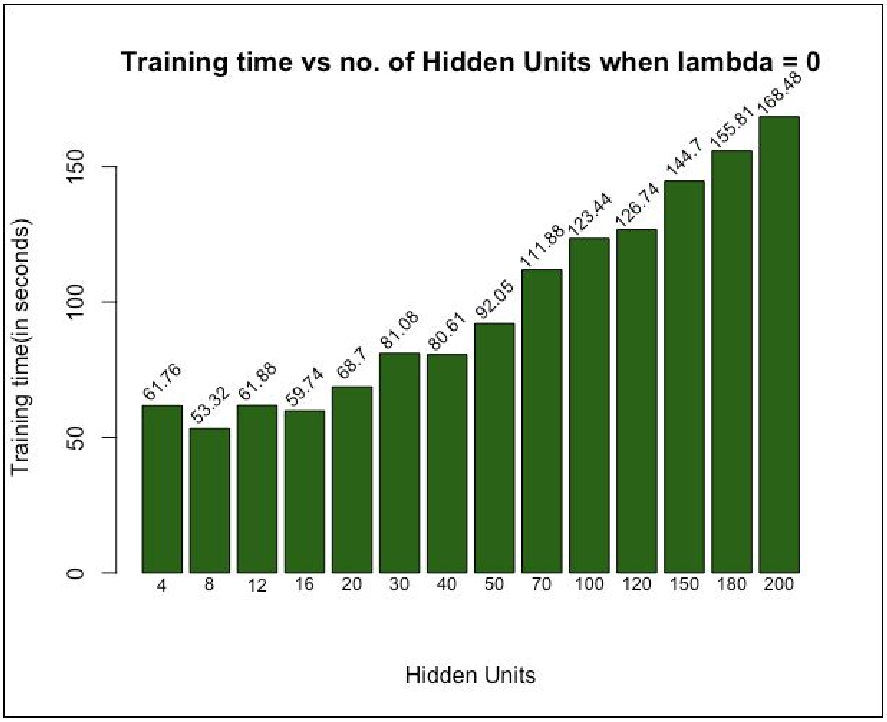
Hidden units	Accuracy when Lambda = 0				Accuracy when Lambda = 5			
	Train	Validation	Test	Time (in sec)	Train	Validation	Test	Time (in sec)
4	65.18	65.95	65.47	61.76	65.27	66.29	64.87	52.38
8	81.66	82.84	82.43	53.32	87.71	88.54	87.39	52.34
12	90.76	91.15	90.56	61.88	91.47	92.12	91.23	56.13
16	93.09	93.27	93.01	59.74	92.75	93.18	92.66	61.21

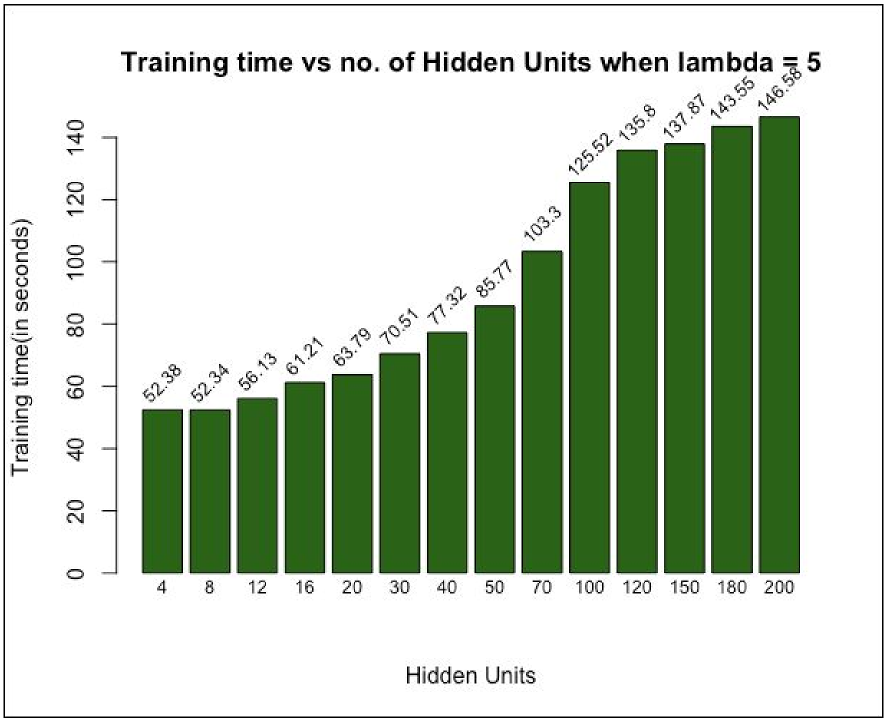
20	92.50	92.98	92.25	68.70	93.19	93.44	92.85	63.79
30	94.72	94.91	94.39	81.08	94.42	94.71	94.14	70.51
40	94.87	95.01	94.65	80.61	94.82	95.01	94.36	77.32
50	95.07	95.20	94.63	92.05	95.20	95.37	94.91	85.77
70	95.47	95.73	95.11	111.88	95.66	95.77	95.41	103.30
100	95.77	95.81	95.55	123.44	95.62	95.78	95.25	125.52
120	**95.54**	**95.79**	**95.34**	**126.74**	**95.66**	**95.94**	**95.54**	**135.80**
150	95.02	95.22	94.82	144.70	94.08	94.72	94.15	137.87
180	95.31	95.58	95.24	155.81	91.39	92.37	91.97	143.55
200	94.26	94.80	94.28	168.48	91.74	92.63	92.36	146.58

We started with small number of hidden units and gradually increased them and later on, increased the interval between them. We observed that the accuracy starts with minimum, increases till a certain point and then decreases after a while. We can observe that for **optimal lambda**, the accuracy is the highest when hidden nodes are **120.** The highest accuracy we get is **95.54%.**

3. Effect on training time

Increasing the number of hidden units resulted an increase in training time of the neural network. Our observations are as follows :

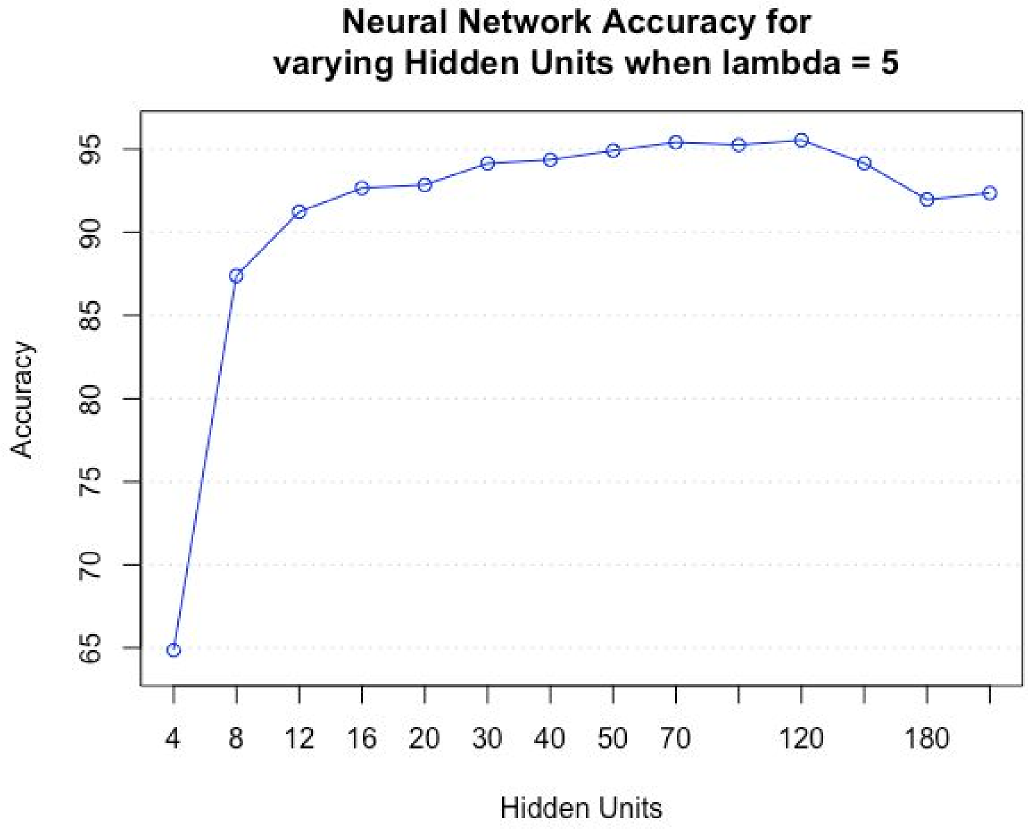

Training time vs no. of Hidden Units when lambda = 0
Training time(in seconds)
0
50
100
150
61.76
53.32
61.88
59.74
68.7
81.08
80.61
92.05
111.88
123.44
126.74
144.7
155.81
168.48
4
8
12
16
20
30
40
50
70
100
120
150
180
200
Hidden Units


Training time vs no. of Hidden Units when lambda = 5
Training time(in seconds)
0
20
40
60
80
100
120
140
52.38
52.34
56.13
61.21
63.79
70.51
77.32
85.77
103.3
125.52
135.8
137.87
143.55
146.58
4
8
12
16
20
30
40
50
70
100
120
150
180
200
Hidden Units

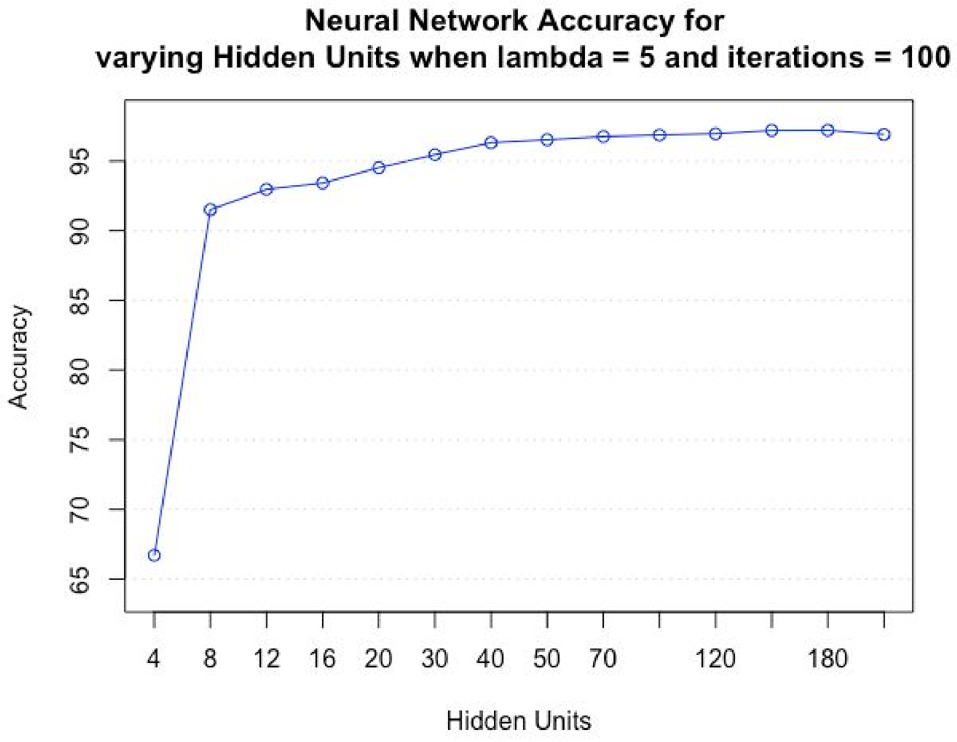
Conclusion:
Thus, we can conclude that **increasing the number of hidden units causes increase in the training time of the neural network**.
Based on the above analysis and conclusion, we can estimate the optimal hyper-parameters for our network are lambda = 5 and number of hidden units = 120.

2.2. Accuracy of classification method on the handwritten digits test data

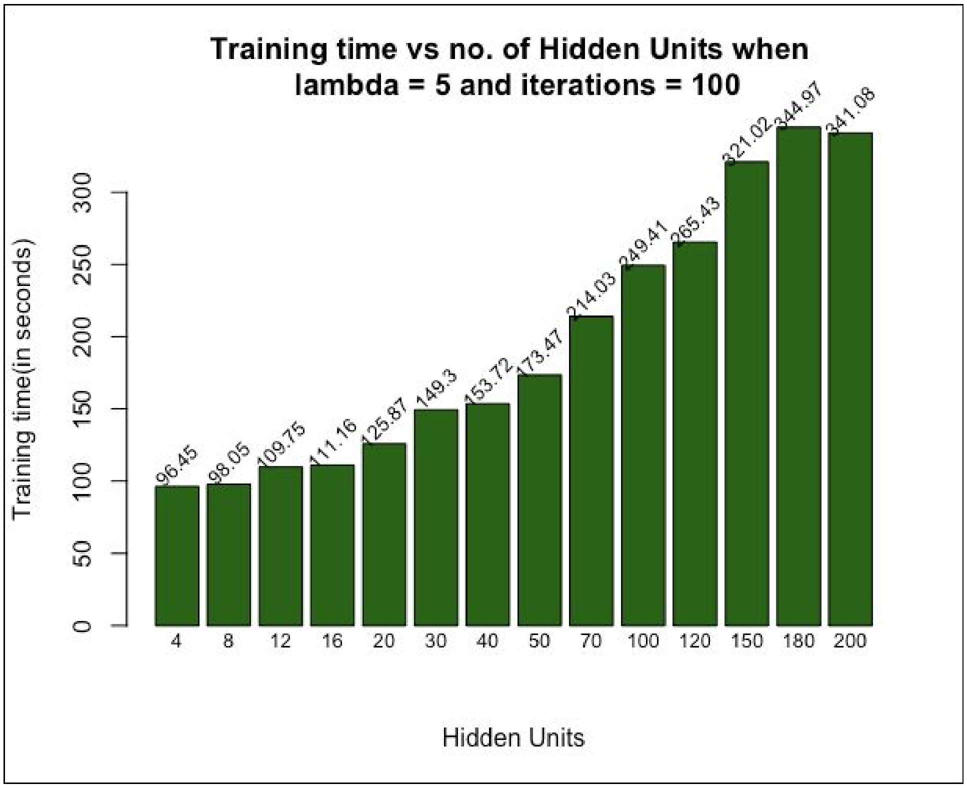
The accuracy of our network using optimal parameters is **95.54%** for 50 iterations :



We also tested our network for **100 iterations**. The accuracy we got is **96.96%** for 120 hidden units and optimal lambda :



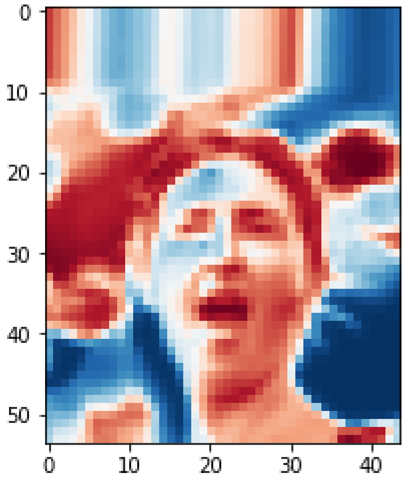
The time taken for **100 iterations** increased drastically with the number of hidden units:



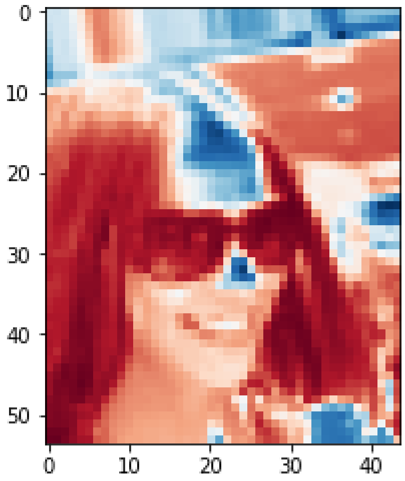
Conclusion:
For 100 iterations, we got accuracy as **96.96%** on test data, when we used the optimal parameters. Increasing the hidden units even more results in overfitting, which is the reason we chose number of hidden units as 120. We cycled through likely values for the parameters (hidden units and lambda constant) in different combinations and assessing some measure of accuracy / fitness for each combination on the validation set. We then selected the best set of parameter values and ran it on the test set.

2.3. Accuracy of classification method on the CelebA dataset

We evaluated the accuracy of our neural network on CelebA dataset using facennScript. This dataset has over 26000 records classified into two class.
Sample class 0 data -



Sample class 1 data -

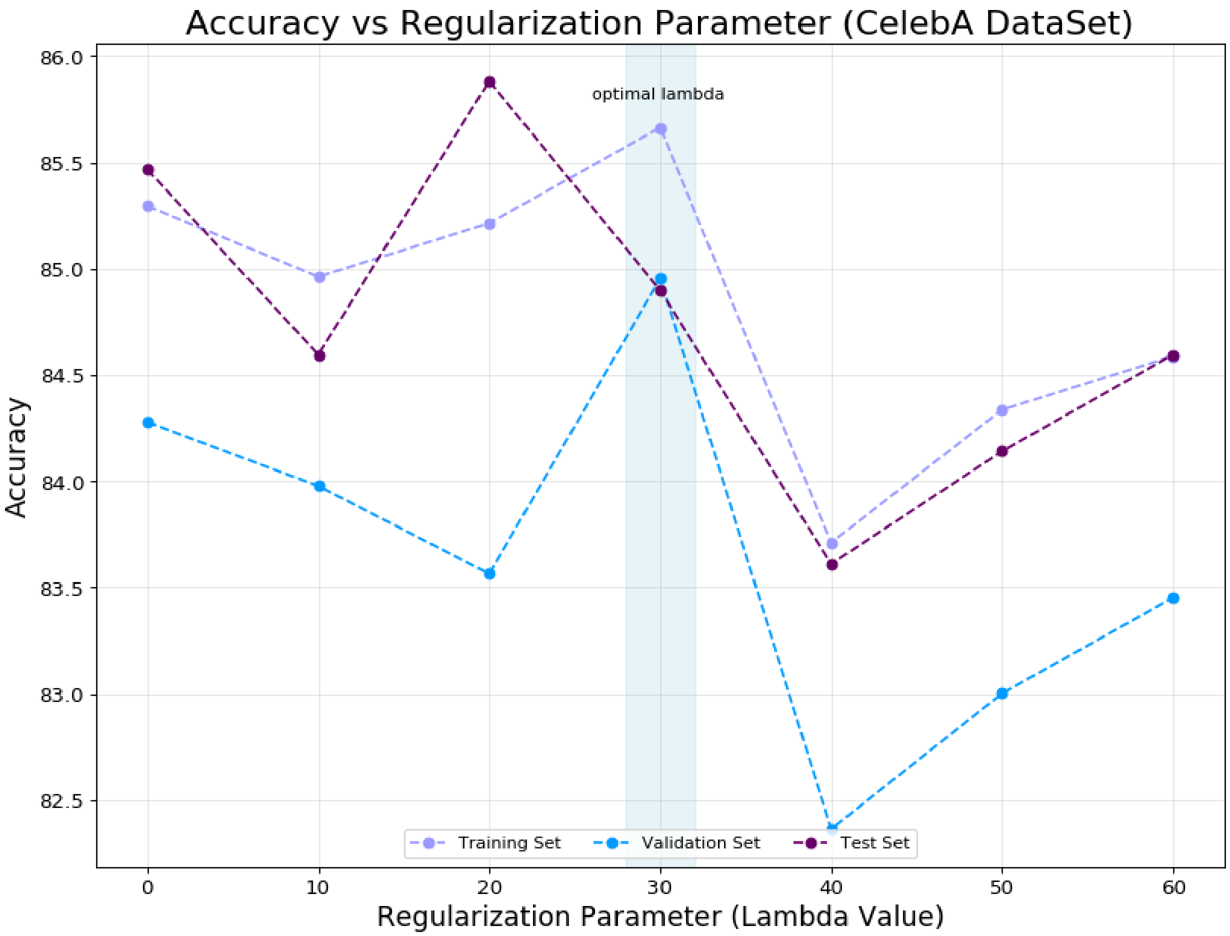


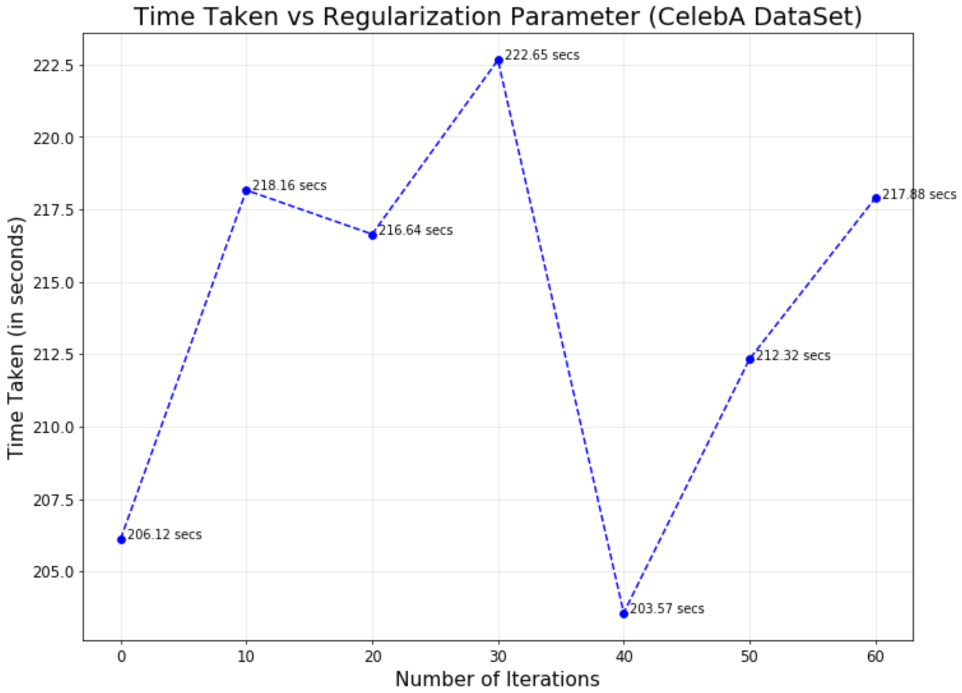
Initially, we tested for different lambda values by keeping the number of iterations same.

The following observations were made for hyperparameter for regularization keeping number of iterations constant at 50.

Regularization Parameter (Lambda Value)	**Accuracy (%)**			**Time Taken (in Seconds)**
	Training Set	**Validation Set**	**Test Set**	
0	85.2938	84.2777	85.4656	206.12
10	84.9621	83.9775	84.5950	218.16
20	85.2133	83.5647	85.8819	216.64
30	**85.6635**	**84.9531**	**84.8978**	**222.65**
40	83.7109	82.3640	83.6109	203.57
50	84.3365	83.0019	84.1408	212.32
60	84.5877	83.4522	84.5950	217.88

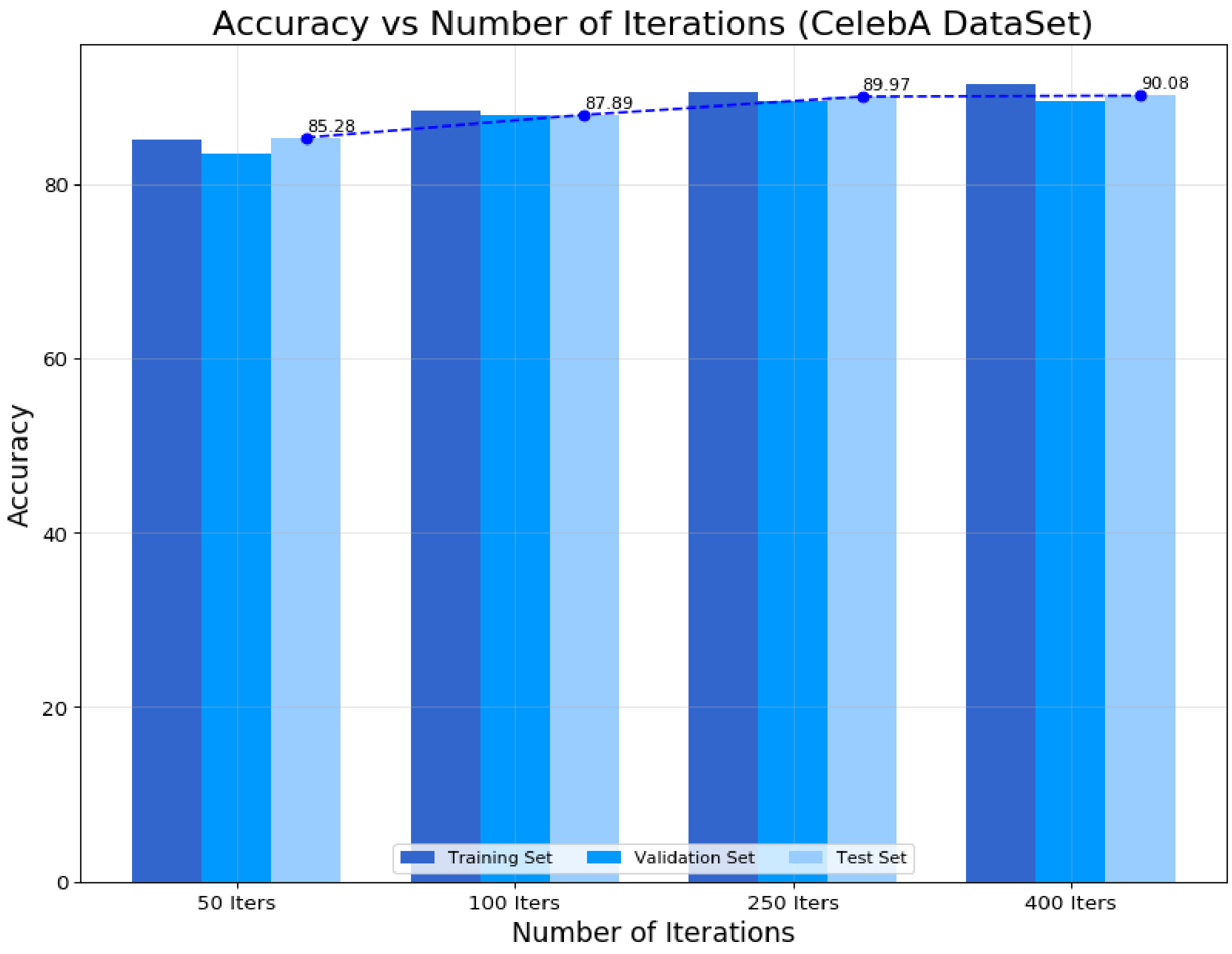
We use our validation data to tune our hyper parameter for regularization. Here we observe that model performs good for regularization parameter 30. Time taken to perform this operation was also considerably more. Once we set our regularization parameter, we can further optimize our cost function using our validation data to improve our accuracy against the testing data. As the parameter increases from the optimal value, model starts giving preference to the weights rather than features which again results in increase of misclassification which lowers the accuracy. Following is the graphical representation of our observation.

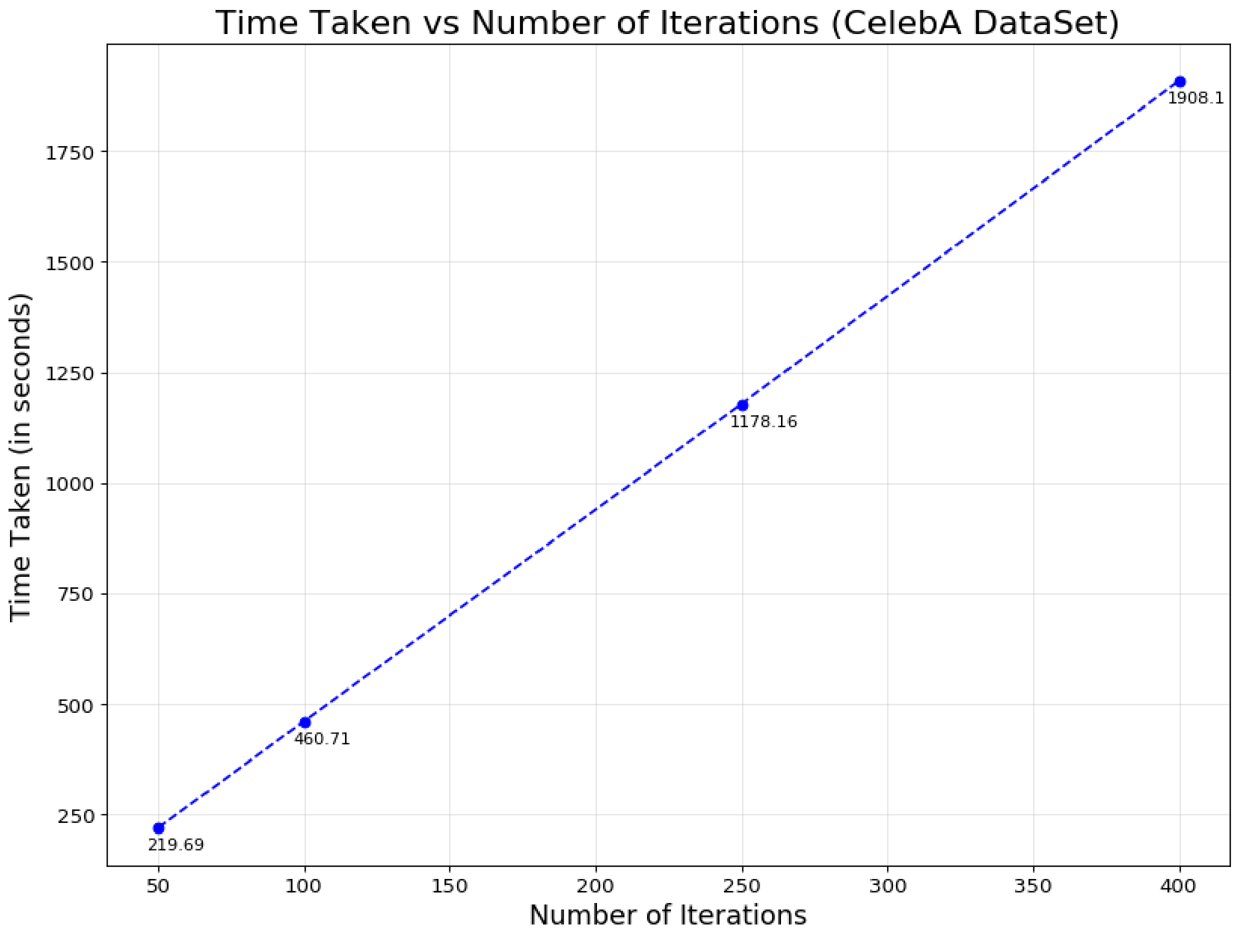

Accuracy vs Regularization Parameter (CelebA DataSet)
optimal lambda
Accuracy
86.0
85.5
85.0
84.5
84.0
83.5
83.0
82.5
0
10
20
30
40
50
60
Regularization Parameter (Lambda Value)
Training Set
Validation Set
Test Set


Time Taken vs Regularization Parameter (CelebA DataSet)
Time Taken (in seconds)
222.5
220.0
217.5
215.0
212.5
210.0
207.5
205.0
206.12 secs
218.16 secs
216.64 secs
222.65 secs
203.57 secs
212.32 secs
217.88 secs
0
10
20
30
40
50
60
Number of Iterations

The comparison of accuracy with different number of iterations with optimal lambda is as follows :

Number of Iterations	Accuracy (%)			Time Taken (in Seconds)
	Training Set	Validation Set	Test Set	
50	84.9526	83.4146	85.2763	219.69
100	88.3886	87.8049	87.8880	460.71
250	90.5071	89.5400	89.9700	1178.16
400	91.3744	89.5310	90.0833	1908.10



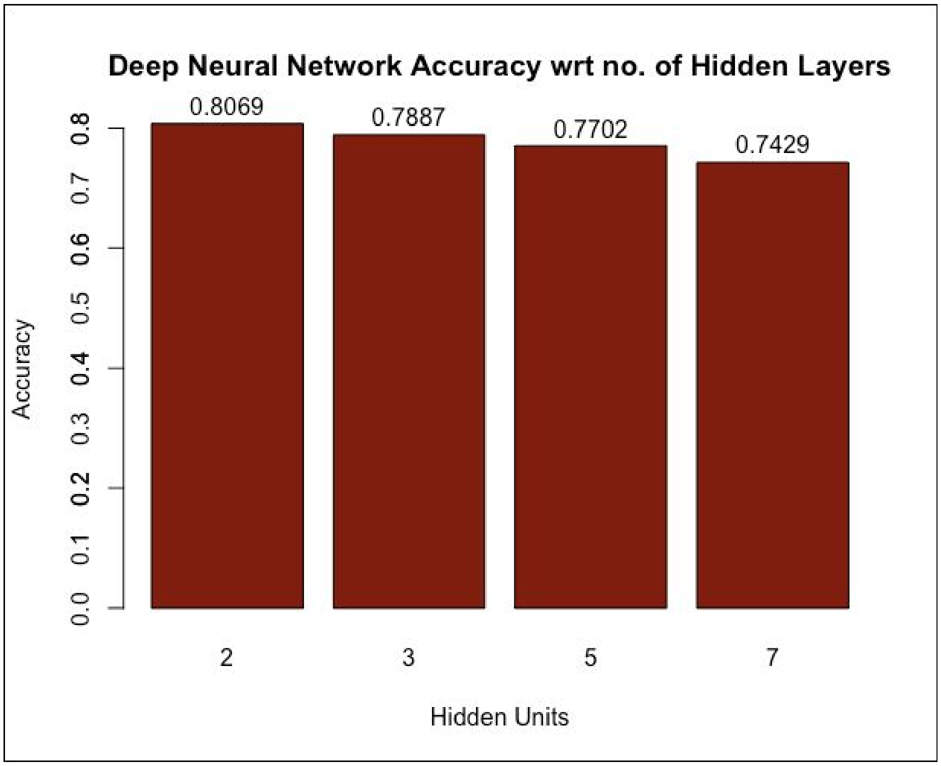


We observe here that as the number of iteration increases time taken to train the model increases linearly. With increase in the number of iterations, performance increases as well. However the rate of increment is not linear.

2.4. Comparison of our Neural Network with Deep Neural Network in terms of accuracy and training time

Deep Neural Network :

We evaluated the accuracy of Deep neural network on CelebA dataset by changing the number of hidden layers. We varied the hidden layers from 3 to 7. The accuracies we got are as follows :

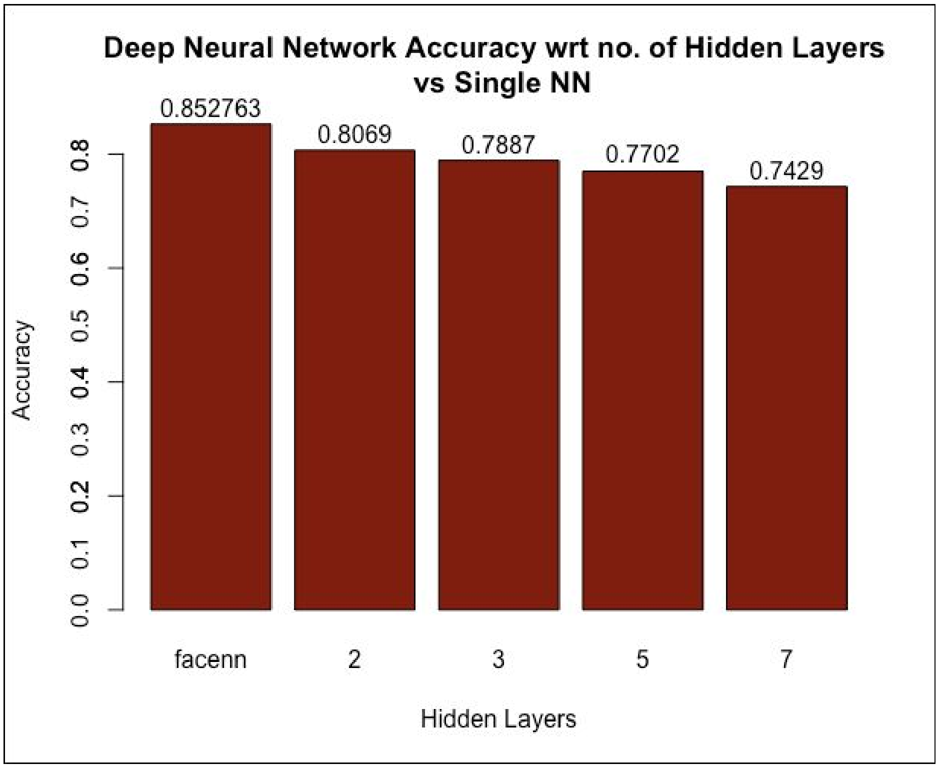


We can observe that the accuracy keeps on **decreasing** as we increase the number of hidden layers.

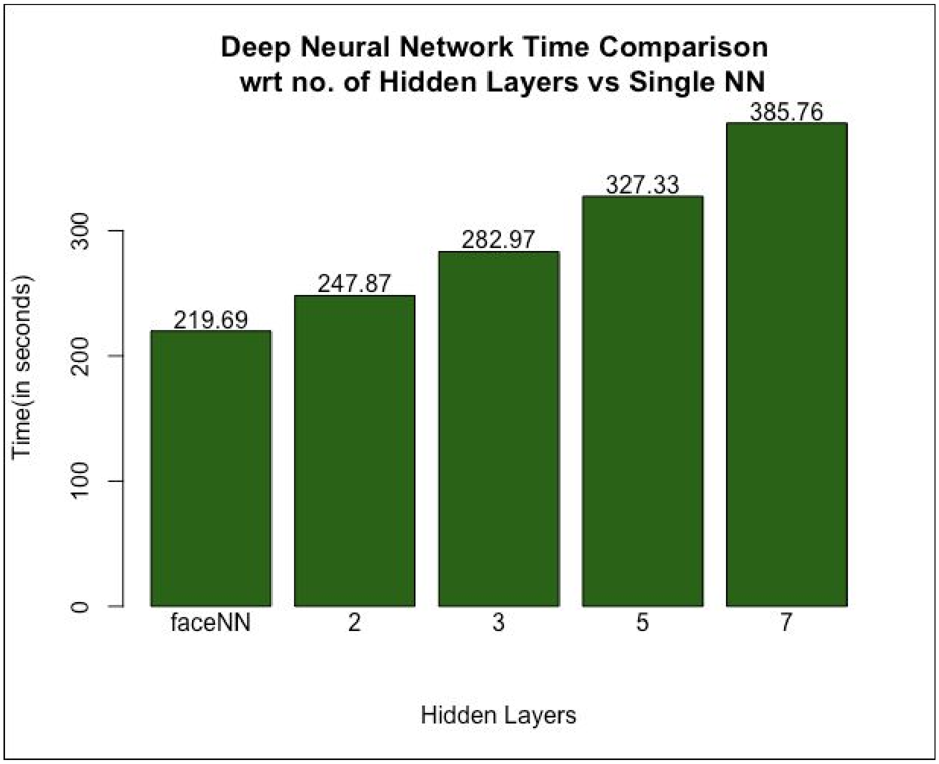
Comparison with our network :

We compared deep neural network for 3,5 and 7 layers with our network trained in faceNN. The comparisons for accuracy and time are as follows:

Accuracy comparison:



Time comparison:



Conclusion :

It can be inferred that **accuracy of single neural network is more** than deep neural network. In fact, the accuracy decreases as we increase the hidden layers. This inference may change by changing the parameters - lambda, number of hidden units/layers and number of iterations. However, for our setup, the above trend is observed.

In terms of training time, it can be observed that facenn script is executed much faster than deepnn script. Also, on **increasing the number of hidden layers, the training time increases**. These observations are based on learning_rate = 0.0001 and training_epochs = 100. Also to note, all our scripts are executed on **Google Collab**.

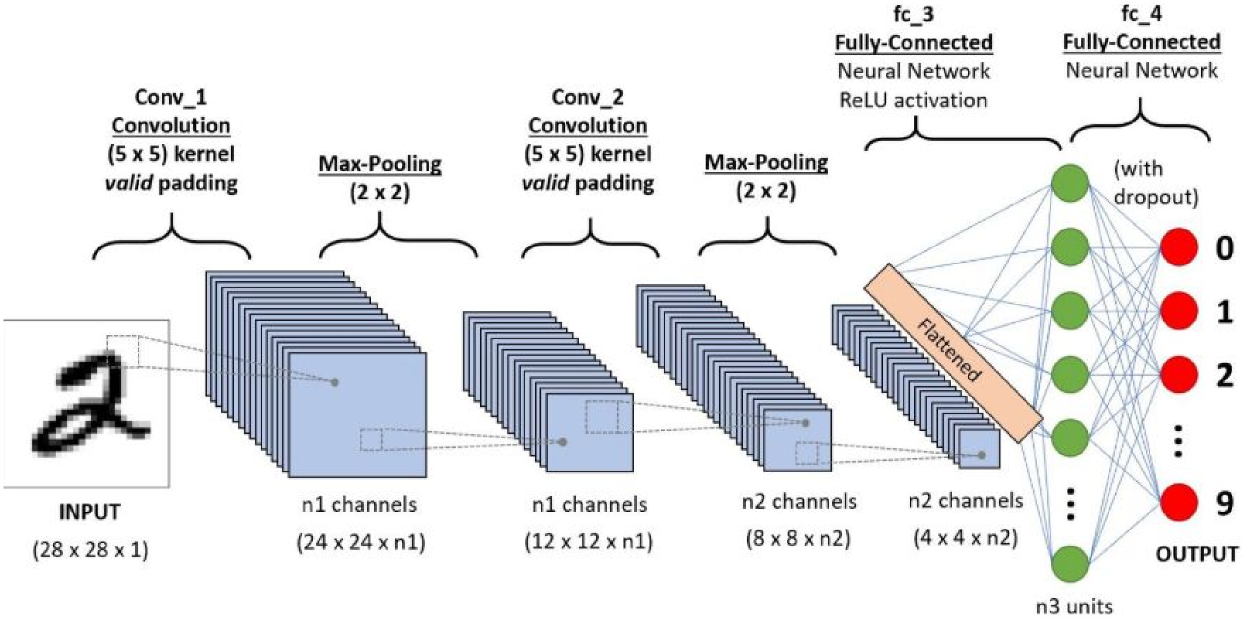
2.5. Results from convolutional neural network in terms of accuracy and training time

Convolutional Neural Network -

In this project, we used the cnnScript for calling the Tensorflow library for a convolutional neural network.Convolutional neural networks ingest and process images as tensors, and tensors are matrices of numbers with additional dimensions. Typically, a CNN is composed of a stack of convolutional modules that perform feature extraction. Each module consists of a convolutional layer followed by a pooling layer. The last convolutional module is followed by one or more dense layers that perform classification. The final dense layer in a CNN contains a single node for each target class in the model (all the possible classes the model may predict), with a **sigmoid** activation function to generate a value between 0–1 for each node (the sum of all these softmax values is equal to 1). We can interpret the softmax values for a given image as relative measurements of how likely it is that the image falls into each target class.

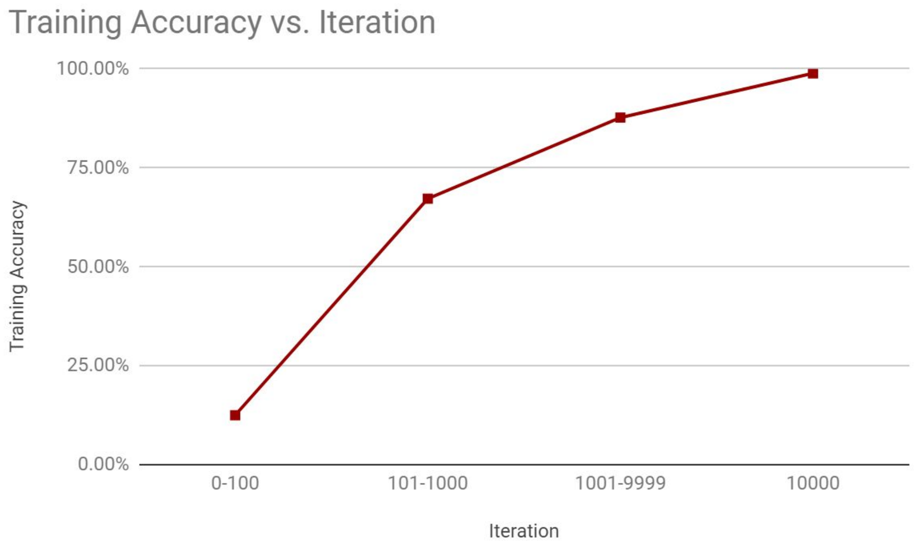
In our Model , we used the below components :

1. **Convolutional Layer #1**: Applies 36 5x5 filters (extracting 5x5-pixel subregions), with ReLU activation function
2. **Pooling Layer** : Performs max pooling with a 2x2 filter and stride of 1
3. **Convolutional Layer #2**: Applies 36 5x5 filters, with ReLU activation function
4. **Pooling Layer #2**: Again, performs max pooling with a 2x2 filter and stride of 1
5. **Dense Layer #1**: Dropout regularization rate of 0.4
6. **Dense Layer #2 (Logits Layer)**: 10 neurons, one for each digit target class (0–9).



Observation :

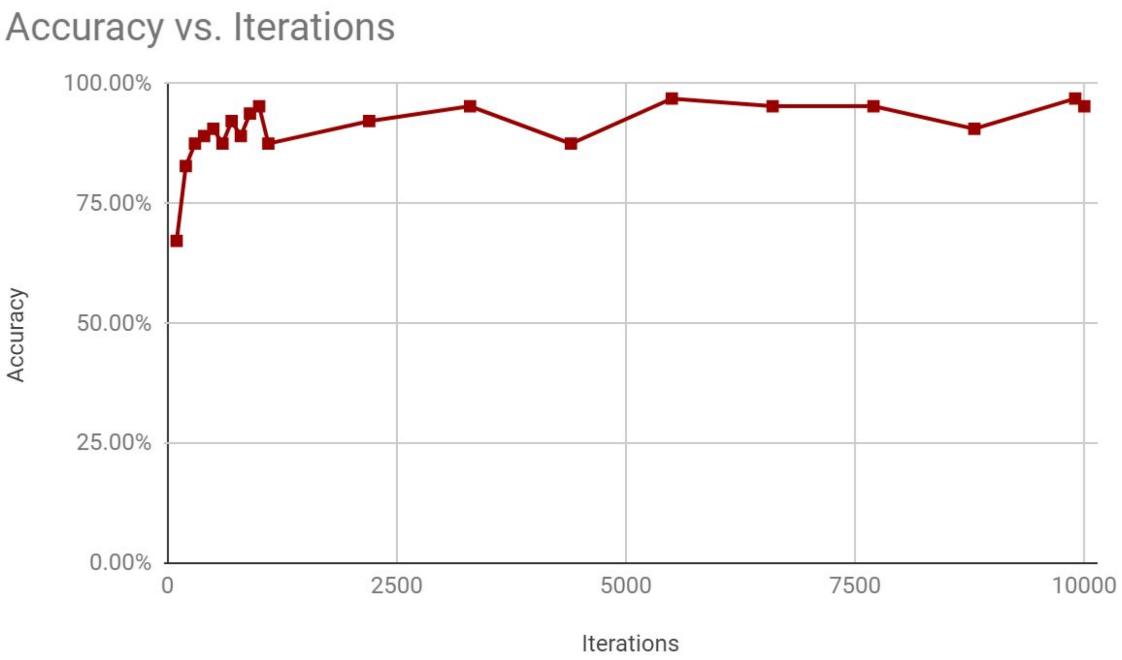
Iteration	Training Accuracy	Accuracy on Test-Set	Time Usage
0	12.5%	9.5% (953 / 10000)	00:00
1-100	67.2%	66.8% (6685 / 10000)	00:08
101-1000	87.7%	94.0% (9401 / 10000)	01:14
10000	98.9%	98.7	12:18(738s)

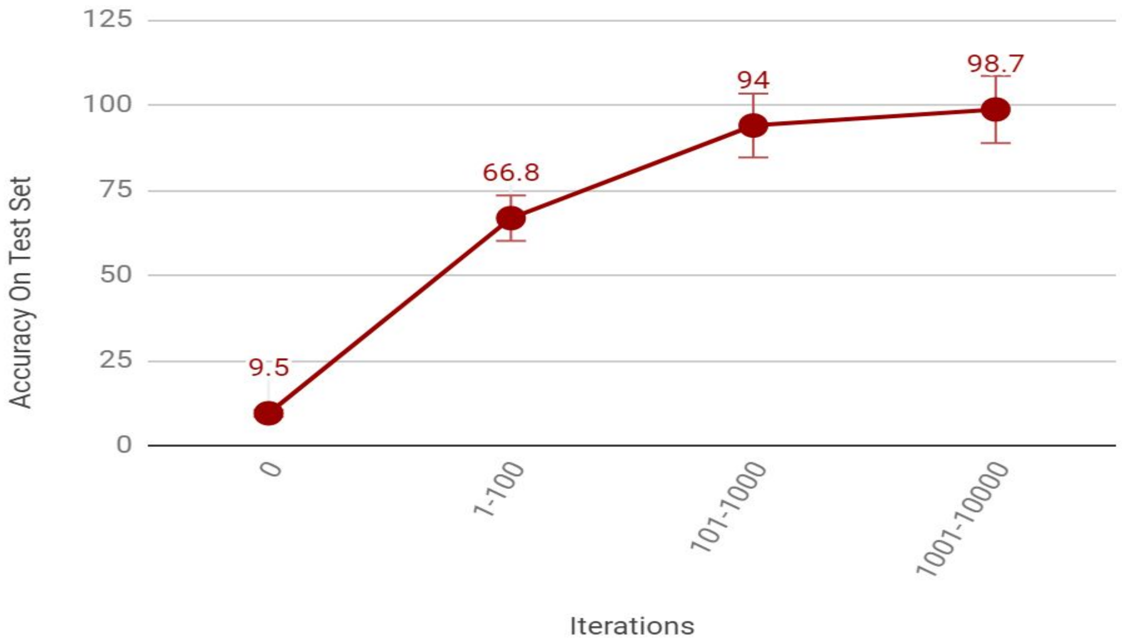


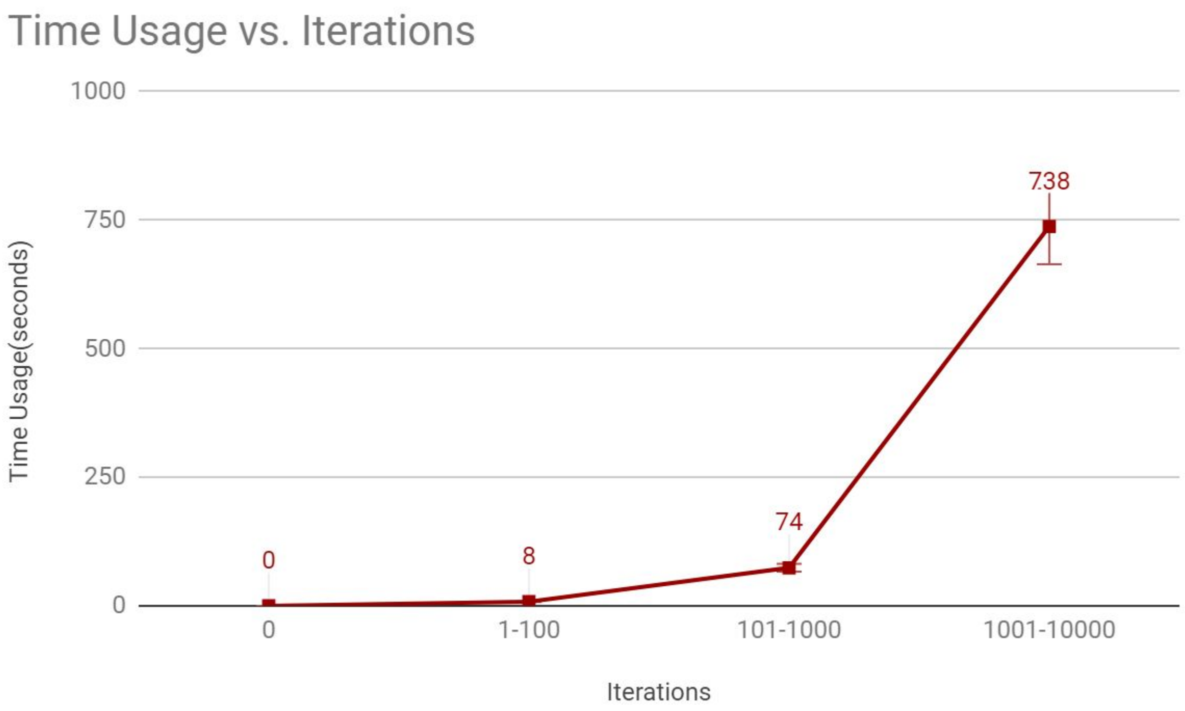
Inference From Graph :

From the above graph it is evident that the Accuracy is directly proportional to the number of Iterations. We repeat the iterations until the accuracy improvement is diminishing and no longer justify the drop in the training and computation performance.From the given data , when we partition the data into sets of iterations of **100, 1000,10000** the data repeatedly recomputes the error and weights due to which the the accuracy plateaus.If after a certain number of epoch, the performance on the test does not improve but stays stagnant, the training stops. This prevents from overfitting and you gain in time of training. From the below graph we can confirm that after a certain point **(above 5000 points mark)** the accuracy almost never changes.

Cumulative Accuracy Over 10000 Iterations :

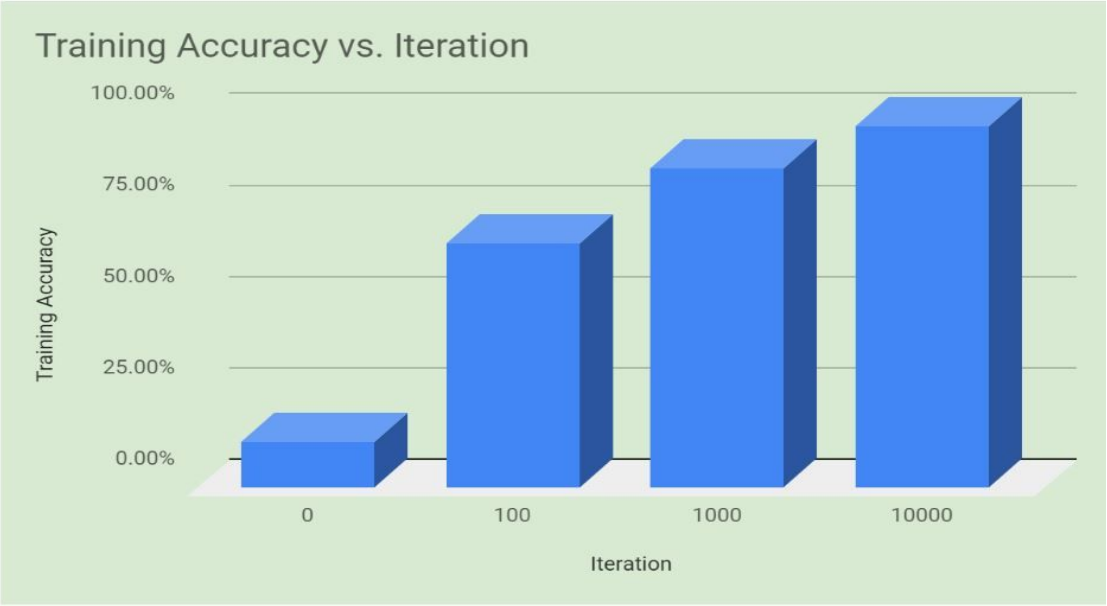

Accuracy vs. Iterations
100.00%
75.00%
50.00%
25.00%
0.00%
Accuracy
0
2500
5000
7500
10000
Iterations


125
100
75
50
25
0
Accuracy On Test Set
9.5
66.8
94
98.7
0
1-100
101-1000
1001-10000
Iterations

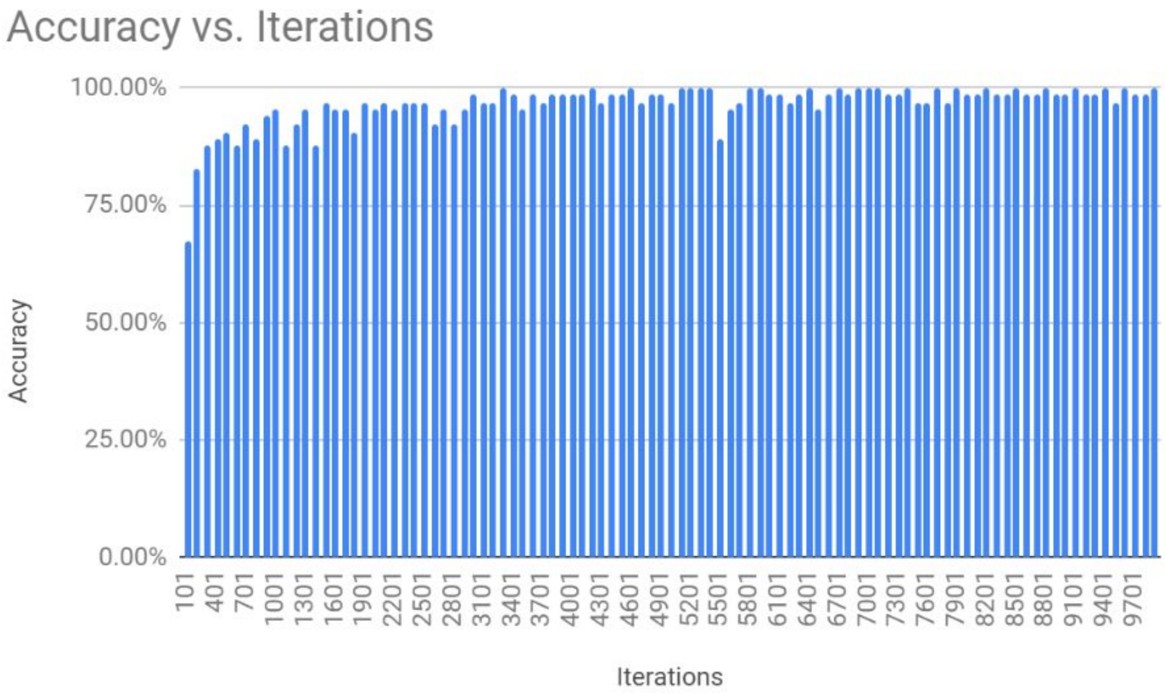


Inference from the graph :

From the above graph we can see that the time needed to train the convolutional neural network increases with the number of iterations. This is expected as the number of iterations increase , the recalculation of **weights** in the Epoch due to which the time taken to train the model increases. The number of iterations is directly proportional to the time needed to train the data.With each iteration would process 10 images for a total of 100 such iterations to go over the entire set. This is just one epoch. Training can go on for ~100s of epochs.

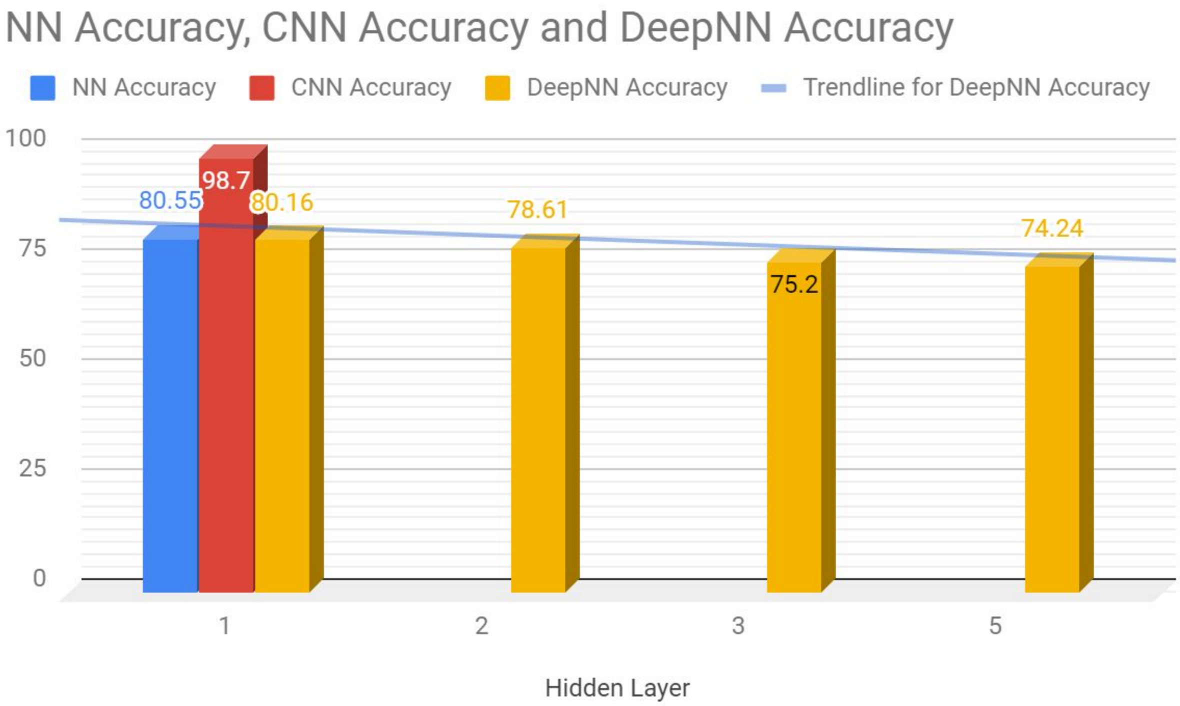


Cumulative Change in the Accuracy over 10000 iterations.

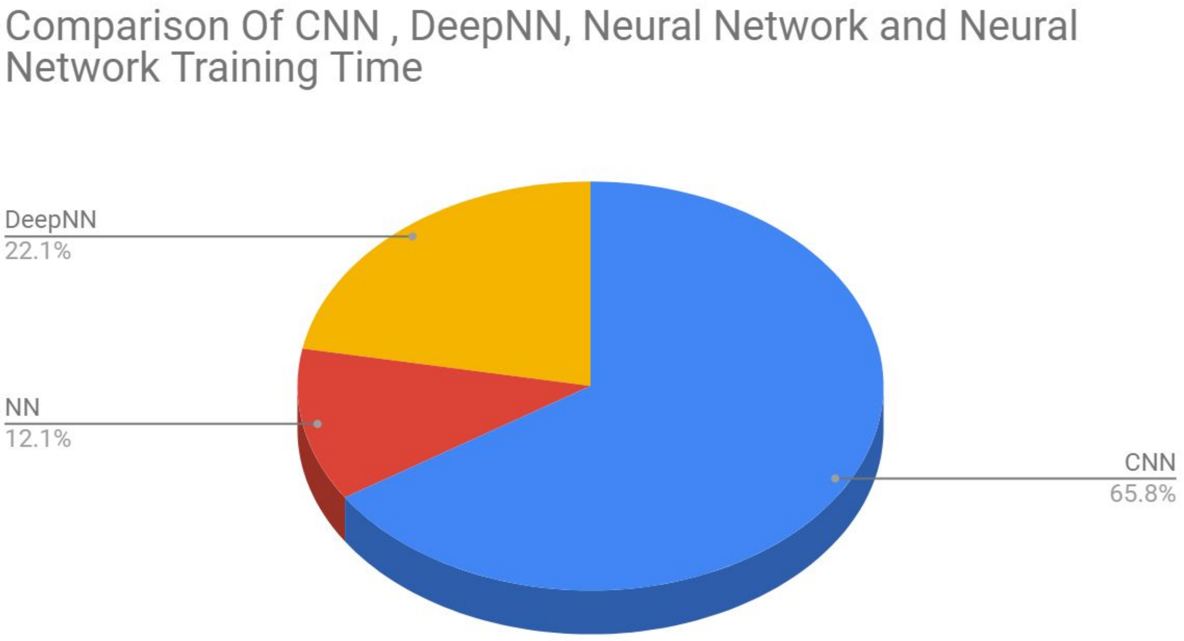


Performance of a Neural Network in comparison with a Convoluted Neural Network :

From the below graph we compared the accuracy of our DeepNN, CNN and Neural Network Model. As expected we can see that the CNN Accuracy is pretty high **(98.7%)** This is because convolutional neural networks reduce the number of units in the network so there are fewer parameters to learn which reduces the chances of overfitting as the model would be less complex than a fully connected neural network.



As Per below graph, the training times for CNN, NN and DeepNN are compared. The training time for **CNN is greater** as there is a forward and backward propagation for each Epoch.

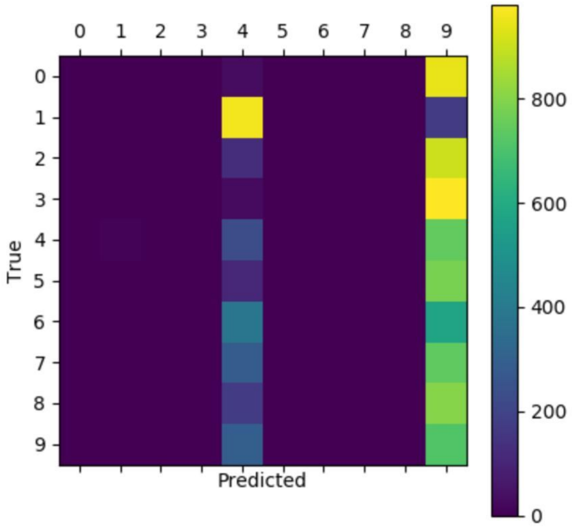


Evaluation Reports from Convolutional Neural Network :

Confusion Matrix is a performance measurement for machine learning classification.**Accuracy** is **calculated** as the total number of two correct predictions (TP + TN) divided by the total number of a dataset (P + N).

```
Accuracy on Test-Set: 8.8% (877 / 10000)
Optimization Iteration:      1, Training Accuracy:   9.4%
Time usage: 0:00:00
```

```
Confusion Matrix:
 [[  0   0   0   0  31   0   0   0   0 949]
 [  0   0   0   0 961   0   0   0   0 174]
 [  0   0   1   0 127   0   0   0   0 904]
 [  0   0   0   0  30   0   0   0   0 980]
 [  0   8   0   0 230   0   0   0   0 744]
 [  0   0   0   0 108   0   0   0   0 784]
 [  0   3   0   0 383   0   0   0   0 572]
 [  0   0   0   0 286   0   0   0   0 742]
 [  0   1   0   0 171   0   0   0   0 802]
 [  0   0   0   0 295   0   0   0   0 714]]
```



```
Optimization Iteration: 1, Training Accuracy:  12.5%
Time usage: 0:00:00
Accuracy on Test-Set: 9.6% (960 / 10000)
```

Confusion Matrix:

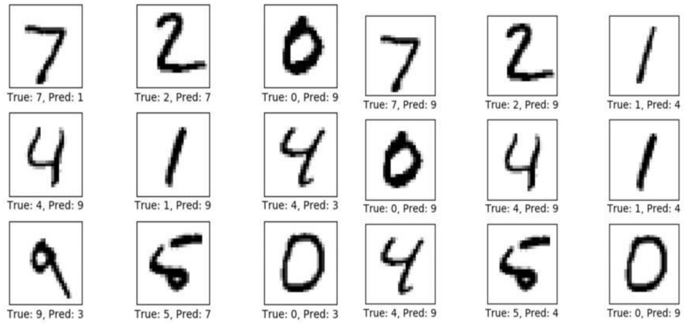
```
[[   0   0      0    0    0    0  980      0    0    0]
 [   0    0   0        0    0    0 1135     0    0    0]
 [   0    0   0        0    0    0 1032     0    0    0]
 [   0    3   0        2    0    0  993   12    0    0]
 [   0    0   0        0    0    0  982    0    0    0]
 [   0    1   0        0    0    0  891    0    0    0]
 [   0    0   0        0    0    0  958    0    0    0]
 [   0    0   0        3    0    0 1025     0    0    0]
 [   0    0   0        0    0    0  974    0    0    0]
 [   0    0   0        0    0    0 1009    0    0    0]]
```

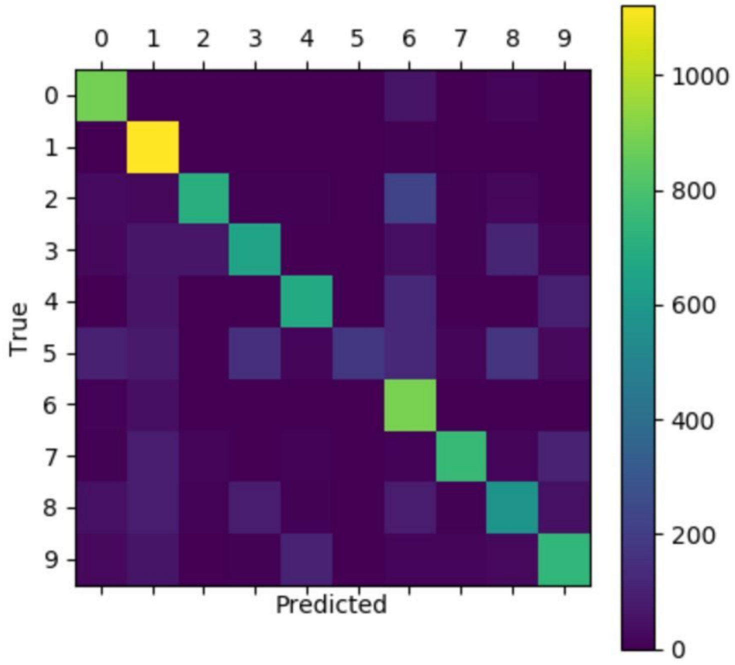
```
Time usage: 0:00:05
Accuracy on Test-Set: 65.4% (6541 / 10000)
```

Confusion Matrix:

```
[[ 946   11    0   12  0     0    4     1    6        0]
 [   0 1119    0   10  0     0        5    0    0        1]
 [  97  162  603 94    4           0      5    1   13   3        5]
 [  40   91   15  831     0     0    2    73  2         1]
 [  26   87    1          0       353    0    78   0           4  4       33]
 [ 291  149  3  287     3      55    17  14 25      48]
 [ 121   91   0          2       5       0    73   8   0       10]
 [  19  124  15 10     50       5    77   3  15      62]
 [  67  227  15 181    2           0      22   21   3   50     89]
 [  65   86    1  14   11           0      3    4    5   11    773]]
```

Error Examples :





```
Time usage: 0:00:56
Accuracy on Test-Set: 93.1% (9307 / 10000)
```

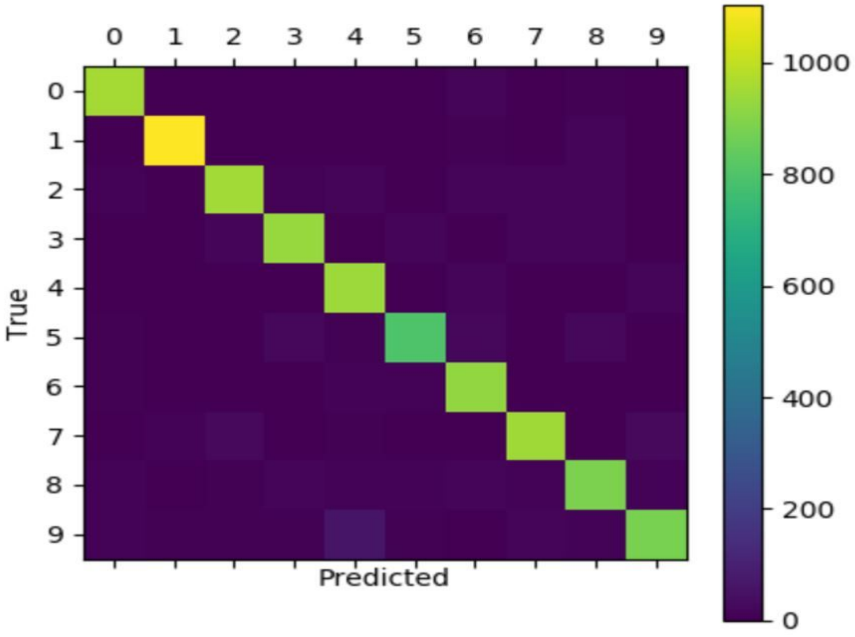
Confusion Matrix:

```
[[ 970   0      2      2      0      3      1      1      1      0]
 [   0 1109     7      3      1      1      5      0      9      0]
 [  16   2    956     14     12      0      5     15     11      1]
 [   2   0     18    942      0     21      0     16      7      4]
 [   2   1      7      1    913      1     11      6      2     38]
 [  14   1      3     30      3    820     11      1      5      4]
 [  17   4      3      1      9     20    901      2      1      0]
 [   2   5     30      4      3      1      0    958      1     24]
 [  14   4 12          40     10     22     10     19    823     20]
 [  14   5      4      14     16      8      0     30      3    915]]
```

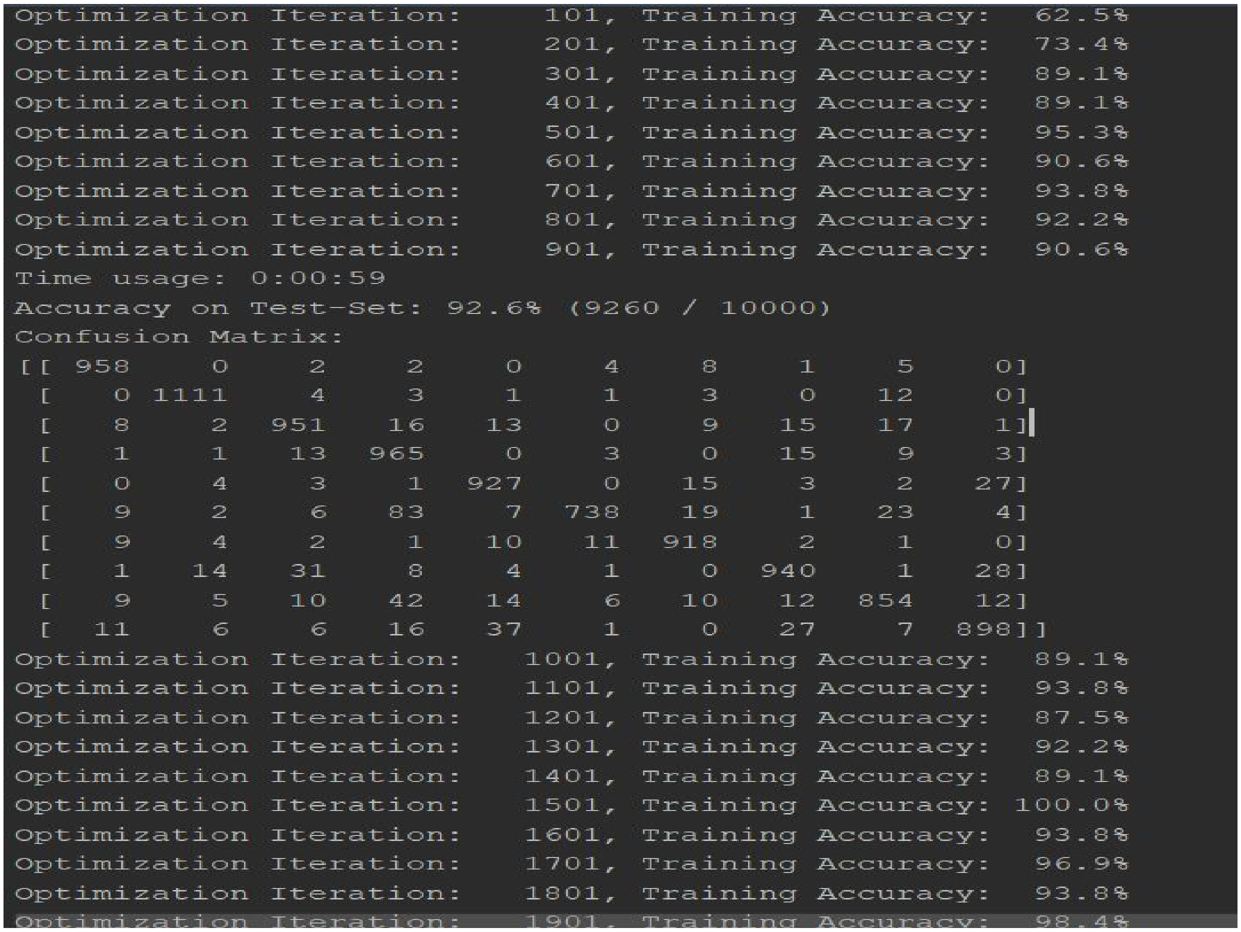
```
Time usage: 0:08:59
Accuracy on Test-Set: 98.7% (9867 / 10000)
```

Confusion Matrix:

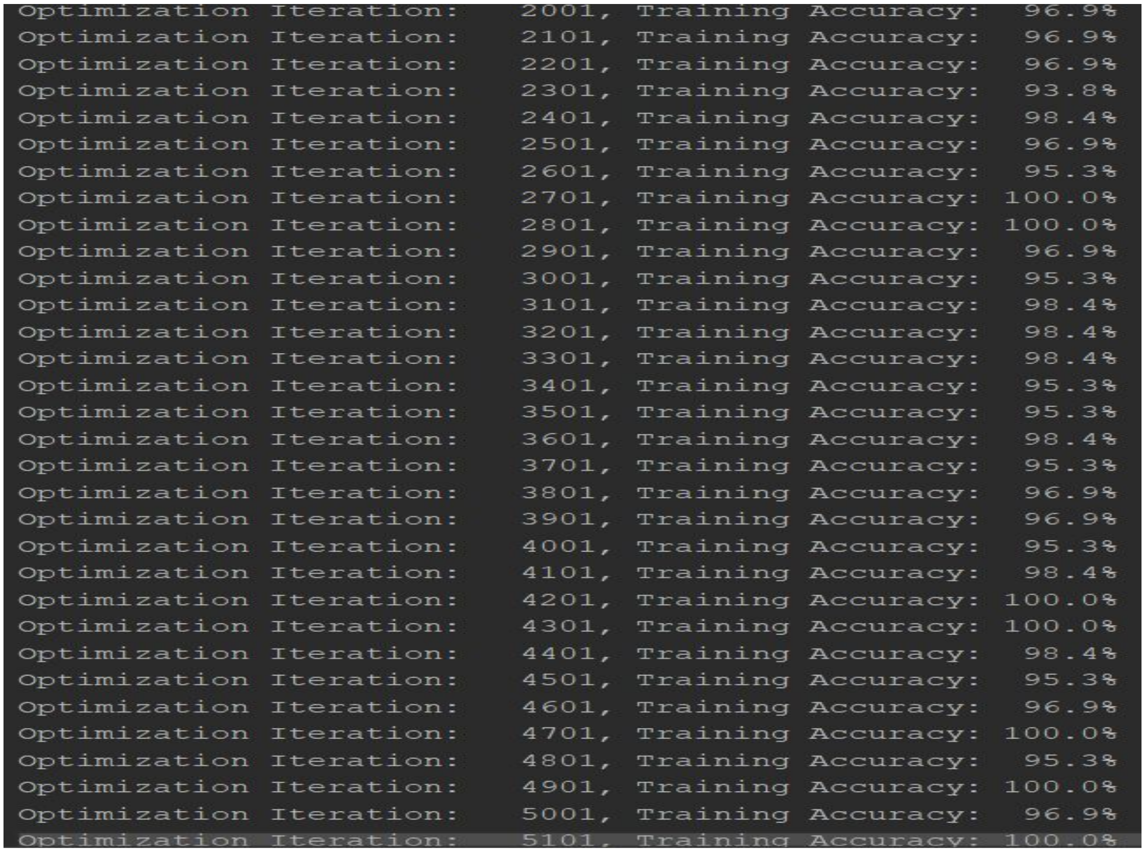
```
[[ 973   0          2        0        0        0        1        1        3        0]
 [   0 1129      2            0        0        0        2        1        1        0]
 [   1      0        1022     1        1        0        0        2        5        0]
 [   0      0        1        999      0        6        0        1        2        1]
 [   0      0        3        0        976      0        1        1        1        0]
 [   2      0        0        5        0        882      1        0        1        1]
 [   3      2        1        1        3        10       936      0        2        0]
 [   0   1           10       2        0        0        0        1013   1           1]
 [   1      0        3        3        1        2        0        2        960      2]
 [   0      4        0        5        12       6        0        4        1        977]]
```



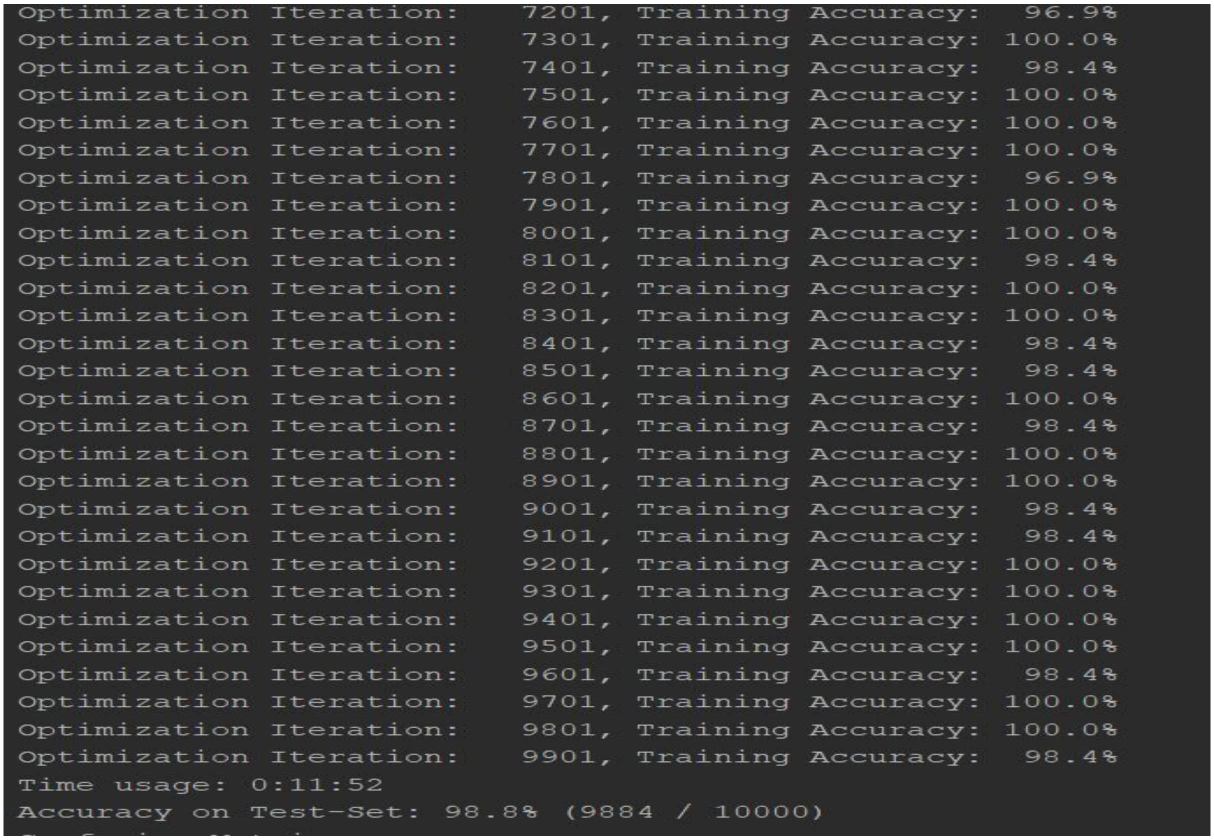
<u>Output : From the Confusion Matrix we are able to deduce the accuracy of the convolutional neural network to be 98.7%</u>



```
Optimization Iteration:    101, Training Accuracy:  62.5%
Optimization Iteration:    201, Training Accuracy:  73.4%
Optimization Iteration:    301, Training Accuracy:  89.1%
Optimization Iteration:    401, Training Accuracy:  89.1%
Optimization Iteration:    501, Training Accuracy:  95.3%
Optimization Iteration:    601, Training Accuracy:  90.6%
Optimization Iteration:    701, Training Accuracy:  93.8%
Optimization Iteration:    801, Training Accuracy:  92.2%
Optimization Iteration:    901, Training Accuracy:  90.6%
Time usage: 0:00:59
Accuracy on Test-Set: 92.6% (9260 / 10000)
Confusion Matrix:
[[ 958    0    2    2    0    4    8    1    5    0]
 [   0 1111    4    3    1    1    3    0   12    0]
 [   8    2  951   16   13    0    9   15   17    1]
 [   1    1   13  965    0    3    0   15    9    3]
 [   0    4    3    1  927    0   15    3    2   27]
 [   9    2    6   83    7  738   19    1   23    4]
 [   9    4    2    1   10   11  918    2    1    0]
 [   1   14   31    8    4    1    0  940    1   28]
 [   9    5   10   42   14    6   10   12  854   12]
 [  11    6    6   16   37    1    0   27    7  898]]
Optimization Iteration:   1001, Training Accuracy:  89.1%
Optimization Iteration:   1101, Training Accuracy:  93.8%
Optimization Iteration:   1201, Training Accuracy:  87.5%
Optimization Iteration:   1301, Training Accuracy:  92.2%
Optimization Iteration:   1401, Training Accuracy:  89.1%
Optimization Iteration:   1501, Training Accuracy: 100.0%
Optimization Iteration:   1601, Training Accuracy:  93.8%
Optimization Iteration:   1701, Training Accuracy:  96.9%
Optimization Iteration:   1801, Training Accuracy:  93.8%
Optimization Iteration:   1901, Training Accuracy:  98.4%
```



```
Optimization Iteration:   2001, Training Accuracy:  96.9%
Optimization Iteration:   2101, Training Accuracy:  96.9%
Optimization Iteration:   2201, Training Accuracy:  96.9%
Optimization Iteration:   2301, Training Accuracy:  93.8%
Optimization Iteration:   2401, Training Accuracy:  98.4%
Optimization Iteration:   2501, Training Accuracy:  96.9%
Optimization Iteration:   2601, Training Accuracy:  95.3%
Optimization Iteration:   2701, Training Accuracy: 100.0%
Optimization Iteration:   2801, Training Accuracy: 100.0%
Optimization Iteration:   2901, Training Accuracy:  96.9%
Optimization Iteration:   3001, Training Accuracy:  95.3%
Optimization Iteration:   3101, Training Accuracy:  98.4%
Optimization Iteration:   3201, Training Accuracy:  98.4%
Optimization Iteration:   3301, Training Accuracy:  98.4%
Optimization Iteration:   3401, Training Accuracy:  95.3%
Optimization Iteration:   3501, Training Accuracy:  95.3%
Optimization Iteration:   3601, Training Accuracy:  98.4%
Optimization Iteration:   3701, Training Accuracy:  95.3%
Optimization Iteration:   3801, Training Accuracy:  96.9%
Optimization Iteration:   3901, Training Accuracy:  96.9%
Optimization Iteration:   4001, Training Accuracy:  95.3%
Optimization Iteration:   4101, Training Accuracy:  98.4%
Optimization Iteration:   4201, Training Accuracy: 100.0%
Optimization Iteration:   4301, Training Accuracy: 100.0%
Optimization Iteration:   4401, Training Accuracy:  98.4%
Optimization Iteration:   4501, Training Accuracy:  95.3%
Optimization Iteration:   4601, Training Accuracy:  96.9%
Optimization Iteration:   4701, Training Accuracy: 100.0%
Optimization Iteration:   4801, Training Accuracy:  95.3%
Optimization Iteration:   4901, Training Accuracy: 100.0%
Optimization Iteration:   5001, Training Accuracy:  96.9%
Optimization Iteration:   5101, Training Accuracy: 100.0%
```



```
Optimization Iteration:   7201, Training Accuracy:  96.9%
Optimization Iteration:   7301, Training Accuracy: 100.0%
Optimization Iteration:   7401, Training Accuracy:  98.4%
Optimization Iteration:   7501, Training Accuracy: 100.0%
Optimization Iteration:   7601, Training Accuracy: 100.0%
Optimization Iteration:   7701, Training Accuracy: 100.0%
Optimization Iteration:   7801, Training Accuracy:  96.9%
Optimization Iteration:   7901, Training Accuracy: 100.0%
Optimization Iteration:   8001, Training Accuracy: 100.0%
Optimization Iteration:   8101, Training Accuracy:  98.4%
Optimization Iteration:   8201, Training Accuracy: 100.0%
Optimization Iteration:   8301, Training Accuracy: 100.0%
Optimization Iteration:   8401, Training Accuracy:  98.4%
Optimization Iteration:   8501, Training Accuracy:  98.4%
Optimization Iteration:   8601, Training Accuracy: 100.0%
Optimization Iteration:   8701, Training Accuracy:  98.4%
Optimization Iteration:   8801, Training Accuracy: 100.0%
Optimization Iteration:   8901, Training Accuracy: 100.0%
Optimization Iteration:   9001, Training Accuracy:  98.4%
Optimization Iteration:   9101, Training Accuracy:  98.4%
Optimization Iteration:   9201, Training Accuracy: 100.0%
Optimization Iteration:   9301, Training Accuracy: 100.0%
Optimization Iteration:   9401, Training Accuracy: 100.0%
Optimization Iteration:   9501, Training Accuracy: 100.0%
Optimization Iteration:   9601, Training Accuracy:  98.4%
Optimization Iteration:   9701, Training Accuracy: 100.0%
Optimization Iteration:   9801, Training Accuracy: 100.0%
Optimization Iteration:   9901, Training Accuracy:  98.4%
Time usage: 0:11:52
Accuracy on Test-Set: 98.8% (9884 / 10000)
```