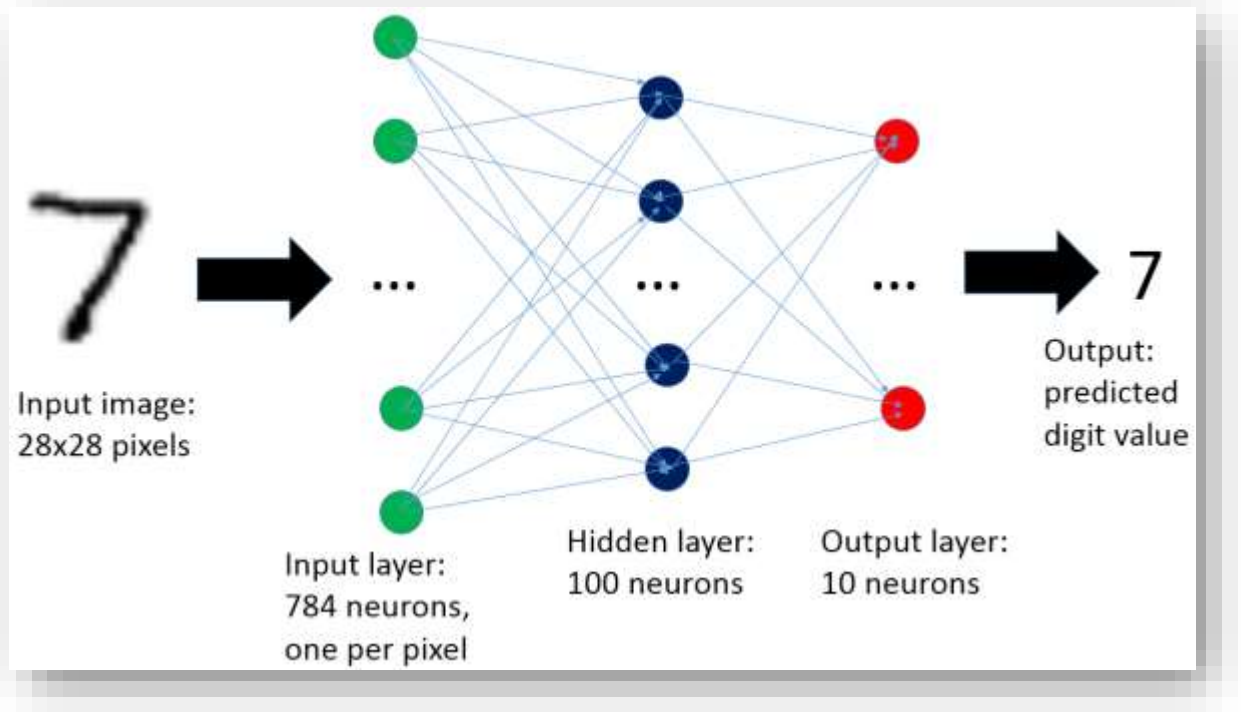
CSE574
Introduction to Machine Learning
Programming Assignment 2
Handwritten Digits Classification

Submitted by:
Divya J Sanghvi -50288057
Anirudh C Ramji -50290168
Saiyam Pravinchandra Shah-50291714
Team No. 78

Introduction:



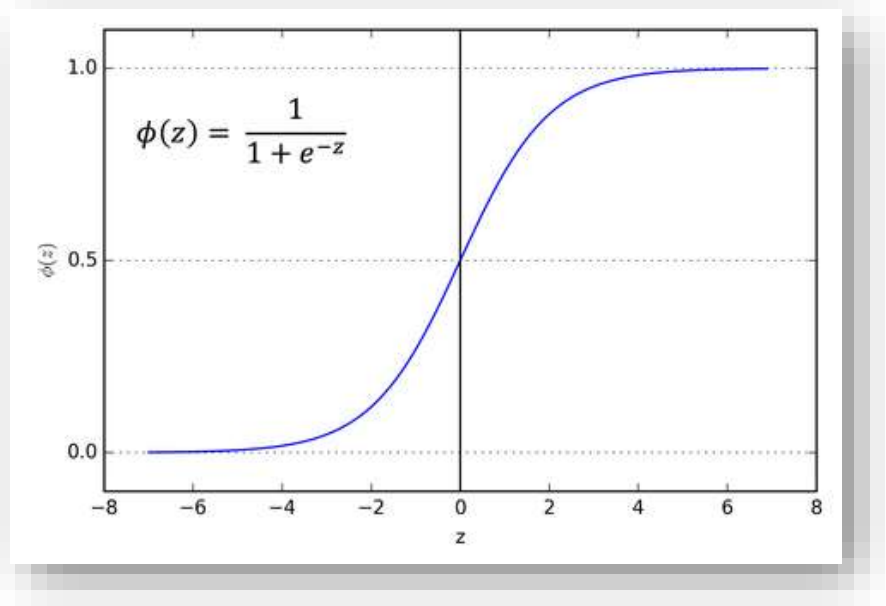
In this assignment, our task was to implement a Multilayer Perceptron Neural Network and evaluate its performance on Handwritten digits classification (MNIST dataset). We also used a Celeb face dataset and compared the accuracy for this dataset for simple neural network versus a deep neural network which was implemented using the TensorFlow Library

Leaning Outcomes:

- How Neural Network works and use Feed Forward, Back Propagation to implement Neural Network?
- How regularization plays a role in the bias-variance tradeoff?
- How to use TensorFlow library to deploy deep neural networks and understand how having multiple hidden layers can improve the performance of the neural network?
- Compare Deep Neural network using TensorFlow vs. Neural Network

1. Implementing Neural Network

a) Sigmoid



b) Preprocess:

We are given 60000 training examples and 10000 testing examples. We split the training set into two parts – 50000 (train set) and 10000 (validation set) For this, on each key we assign 1000 random examples to the validation set. In total there are 10 keys, so we get total of 10000 examples in the validation set and 50000 in the training set. The keys are specified as “train0”,” train1”, “test3”, etc. We use these keys to assign the correct label to the examples

Feature Selection in Preprocessing Step:

In the dataset, we can observe that there are many features which have values that are exactly the same for all data points in the training set. With those features, the classification models cannot gain any more information about the difference (or variation) between data points. Therefore, we can

ignore those features in the pre-processing step. To remove this we remove all the columns that are unnecessary by the following code:

```
# Feature selection
all_cols = np.concatenate((training_set, validation_set, testing_set), axis=0)

del_list = np.all(all_cols == all_cols[0,:], axis=0)
dl = []
for i in range(len(del_list.tolist())):
    if(del_list[i] == True):
        dl.append(i)
testing_set = np.delete(testing_set, dl, axis=1)
validation_set = np.delete(validation_set, dl, axis=1)
training_set = np.delete(training_set, dl, axis=1)
```

We then compare the actual columns to the columns that we obtain after the feature selection and find the result as:

The following pixels were removed after we apply feature selection to our set.

[0,1,2,3,4,5,6,7,8,9,10,11,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,52,53,54,55,56,57,82,83,84,85,111,112,140,168,476,560,644,671,672,673,699,700,701,727,728,729,730,754,755,756,757,758,759,780,781,782,783]

This denotes that these pixels are uninformative and are not needed for our algorithm. One can explain this as it being some random pixels in a picture. When removed, the picture is still able to convey the image that is meant to be. (some grey pixels in between)

c) Implementing nnObj function

In this we did two parts first was a feed forward pass for the neural network and then the backpropagation method in which the weights are learnt by calculating the error and trying to reduce it by adjusting the weights.

d) Prediction

For this step we test our trained models and the weights that we learnt and tested on our training set. For this we just implement a forward pass and compare the output of the 10 output nodes. The maximum output corresponds to our predicted label for our dataset. We use this data to then calculate the accuracies and various other results.

Results

a) We evaluate several results for our model that we have learnt, first being we vary the number of iterations for our neural network and achieve the following best results

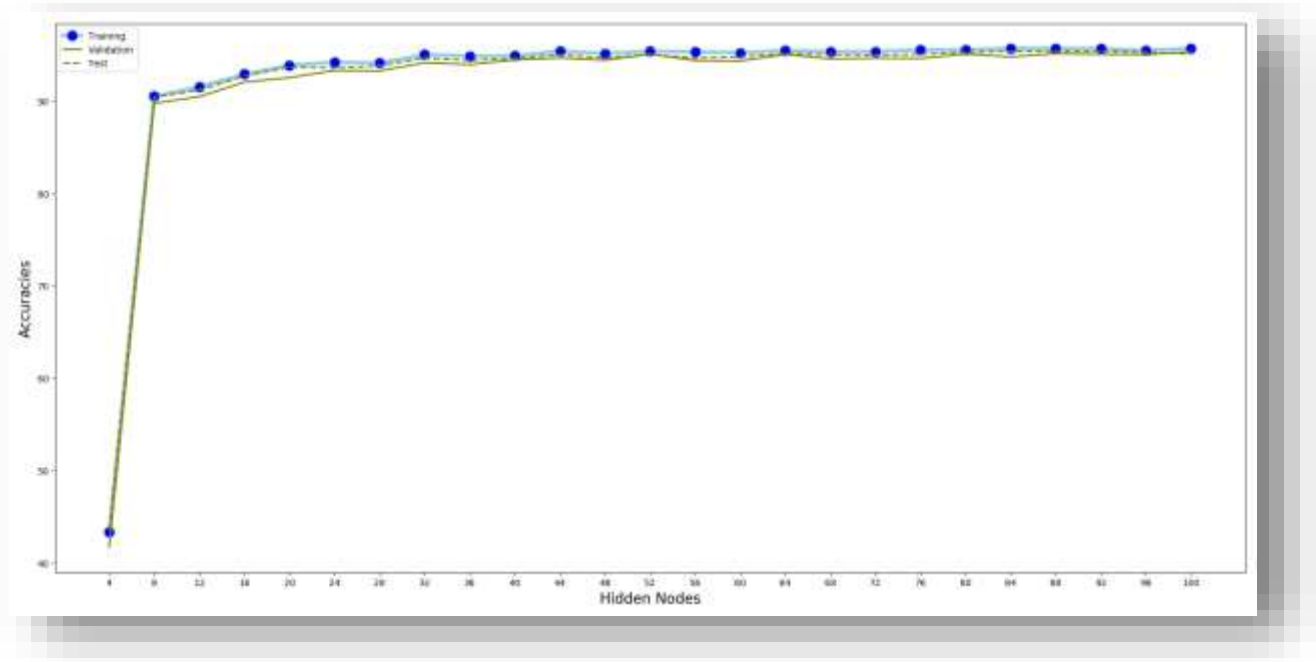
Best Optimum values

Maximum Iterations	Hidden Nodes	Regularization Coefficient	Training Data	Validation Accuracy	Test Accuracy
50	100	5	95.55%	94.58%	95.11%
100	**100**	**5**	**98.26%**	**96.72%**	**97.06%**
150	100	5	98.87%	97.58%	97.50%
200	100	5	99.39%	97.55%	97.92%

We see that as we increase the number of iterations our accuracies increase*, but also, we face the problem of overfitting if we increase the number of iterations. An added negative point is that as the number of iterations keep on increasing the time taken to learn the model increases by a huge amount which we will see later.*

b) Next we evaluate the accuracies of our models by varying the number of hidden nodes. We plotted a graph comparing all three – training, validation, testing accuracies against the number of hidden nodes

Accuracy analysis of Neural Network for handwritten digits:

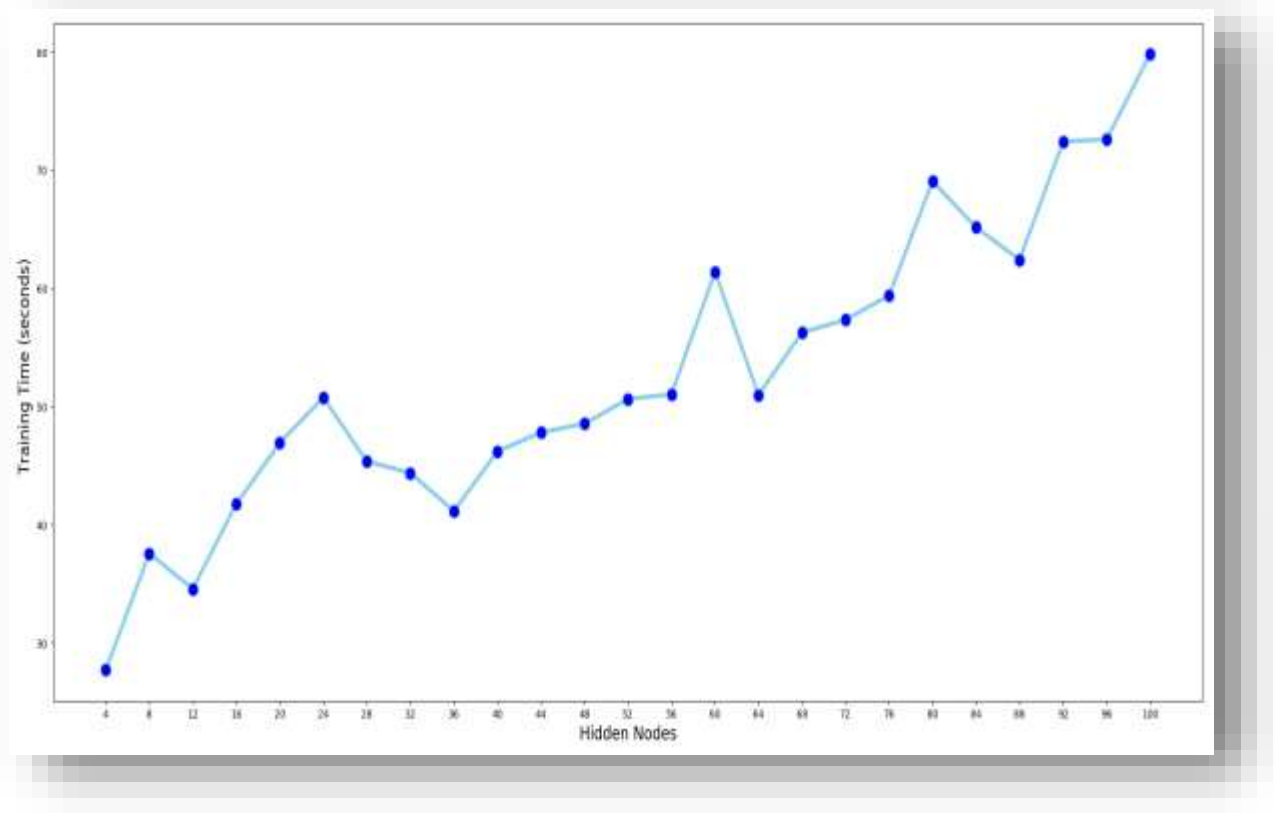



```
training = [43.34, 90.482, 91.498, 92.86999999999999, 93.84599999999999,
            94.158, 94.072, 94.986, 94.834, 94.846, 95.37, 95.106, 95.352,
            95.294, 95.19, 95.422, 95.30999999999999, 95.324, 95.5, 95.514,
            95.666, 95.654, 95.636, 95.47, 95.64399999999999]

validation_list = [41.71, 89.71000000000001, 90.44, 92.0, 92.5, 93.26,
                   93.23, 94.12, 93.91000000000001, 94.42, 94.63000000000001,
                   94.38, 95.04, 94.35, 94.3, 95.00999999999999, 94.53, 94.61, 94.57,
                   95.07, 94.74000000000001, 95.1, 95.03, 95.0, 95.28]

test_list = [43.57, 90.38000000000001, 91.16, 92.65, 93.64, 93.55, 93.77,
             94.63000000000001, 94.35, 94.59, 94.97, 94.64, 95.0, 94.62, 94.75,
             95.08, 95.0, 94.93, 95.05, 95.24000000000001, 95.34, 95.34,
             95.35, 95.24000000000001, 95.06]
```

This denotes that accuracy of our model increases as the number of the hidden nodes increase

c) Next, we evaluate the training time when the number of hidden nodes increase as we discussed earlier

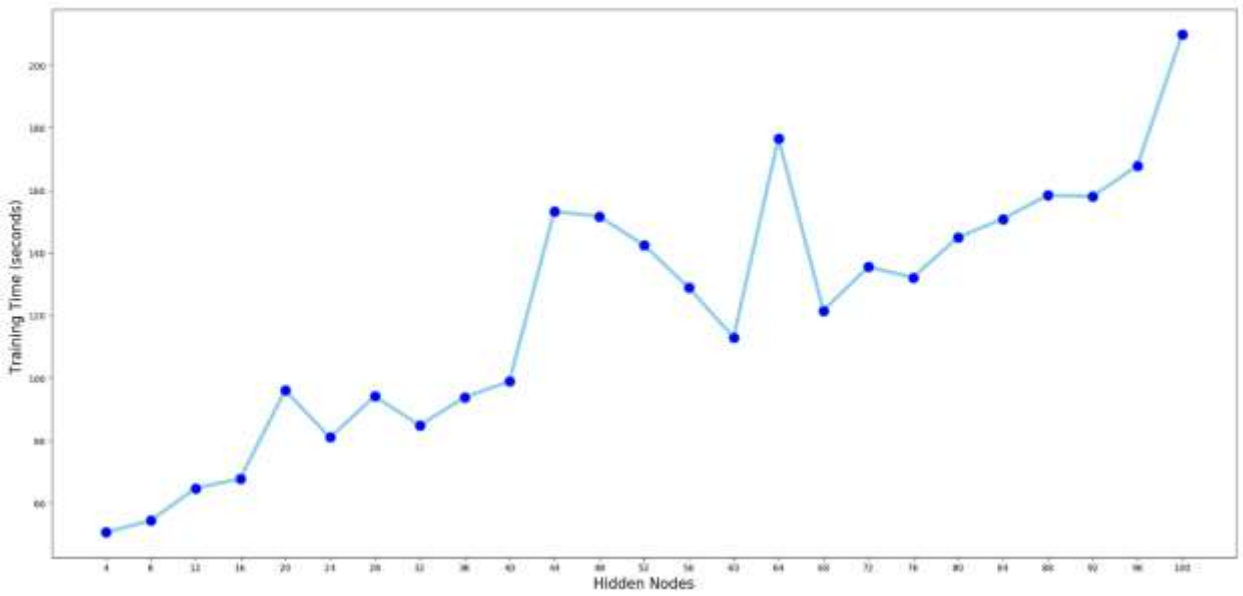
Training Time analysis of Neural Network for handwritten digits



We can clearly see that the training time increases as the number of hidden nodes increase.

For this experiment, we used the number of iterations as 50

Next, we will try for number of iterations 100 and compare these results also.



We can also see that the as the number of iterations increase the training time increases for the hidden nodes

d) Impact of Regularization coefficient on accuracy:

Regularization is an important concept in Machine Learning. Our training set gives pretty high accuracy as it is, but the testing set is not that accurate. This clearly denoted the issue of overfitting; we add the regularization term in our objective function to control the overfitting. It is used to solve the magnitude problem of the weights.
Therefore, our objective function can be rewritten as follow:

$$\widetilde{J}(W^{(1)}, W^{(2)}) = J(W^{(1)}, W^{(2)}) + \frac{\lambda}{2n}\left(\sum_{j=1}^{m}\sum_{p=1}^{d+1}(w_{jp}^{(1)})^2 + \sum_{l=1}^{k}\sum_{j=1}^{m+1}(w_{lj}^{(2)})^2\right)$$

With this new objective function, the partial derivative of new objective function with respect to weight from hidden layer to output layer can be calculated as follow:

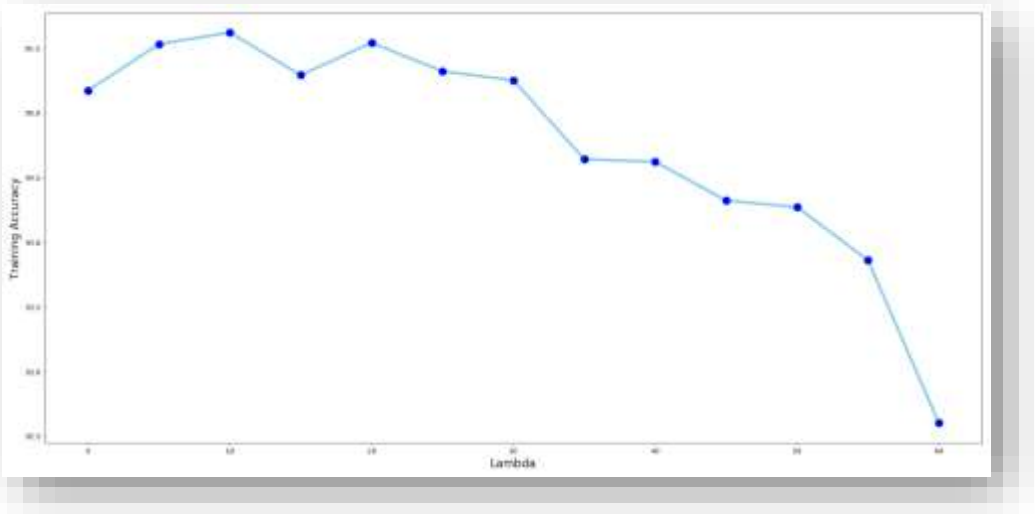
$$\frac{\partial \widetilde{J}}{\partial w_{lj}^{(2)}} = \frac{1}{n}\left(\sum_{i=1}^{n} \frac{\partial J_i}{\partial w_{lj}^{(2)}} + \lambda w_{lj}^{(2)}\right)$$

Similarly, the partial derivative of new objective function with respect to weight from input layer to hidden layer can be calculated as follow:

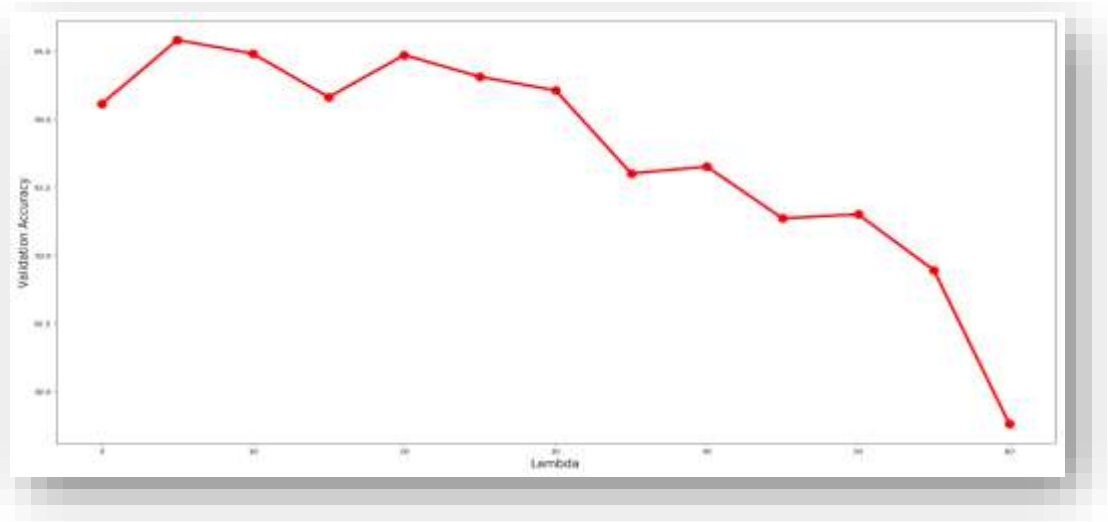
$$\frac{\partial \widetilde{J}}{\partial w_{jp}^{(1)}} = \frac{1}{n}\left(\sum_{i=1}^{n} \frac{\partial J_i}{\partial w_{jp}^{(1)}} + \lambda w_{jp}^{(1)}\right)$$

We plot the graphs, of training vs Lambda, testing vs Lambda and, validation vs lambda and find the results

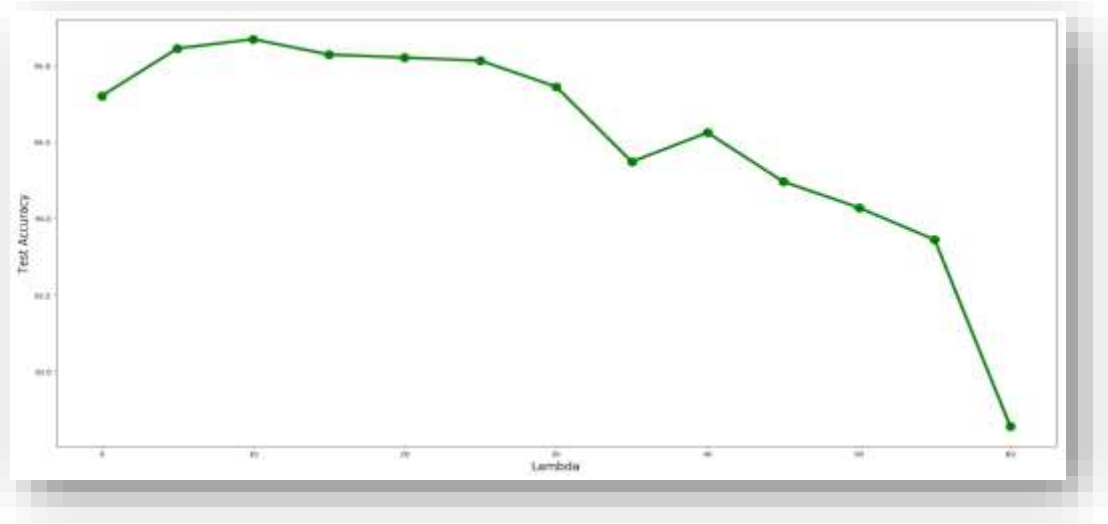
Training Data Set graph vs regularization parameter:



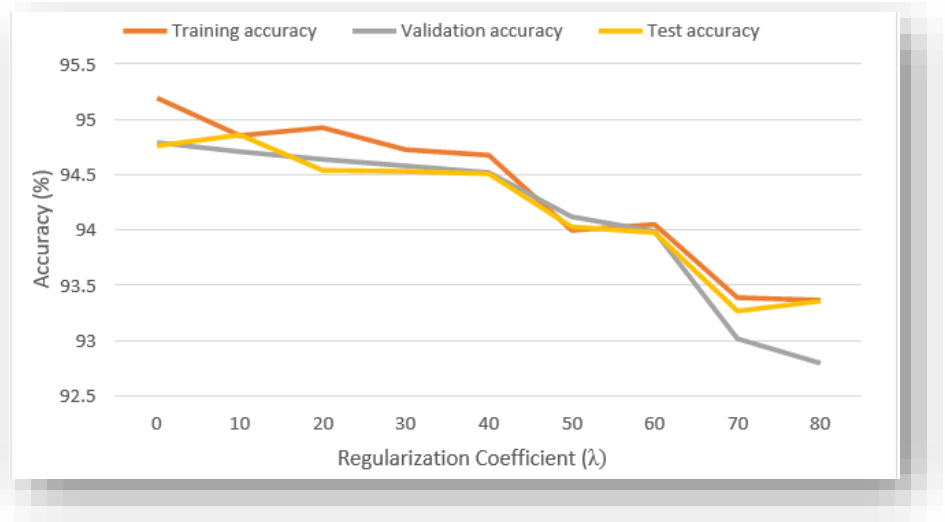
Validation set graph vs regularization parameter:



Testing set graphs vs regularization parameter:



We can see that as the regularization coefficient increases the accuracy decreases for all training, testing and validation, and our ideal value is around 5



2. *Deep Neural Network vs Neural Network*

Comparative Analysis of Single Hidden layer implementation and Deep Neural Network on the CelebA data set

Performance of Implemented algorithm on face Dataset

We used the same forward feed and back propagation algorithm/code for CelebA dataset and below are the observations.

Training set Accuracy	**88.44%**
Validation set Accuracy	**87.84%**
Test set Accuracy	**88.79%**
Execution Time	**170.97s**

Deep Neural Network Performance on face Dataset

Number of Hidden Layers	Execution Time	Accuracy
2	**128.83**	**80.96%**
3	**145.65**	**79.1%**
5	**165.24**	**75.88%**
7	**199.47**	**75.43%**

Comparison

1) **The accuracy of neural network with one layer is better than that of deep neural network**
2) **In terms of training time, deepNN with 2 layers is slightly quicker than that of neural network with 1 layer**

This is because we don't want to overcomplicate the network. DeepNN is faster than Neural Network because the weights are leant faster.

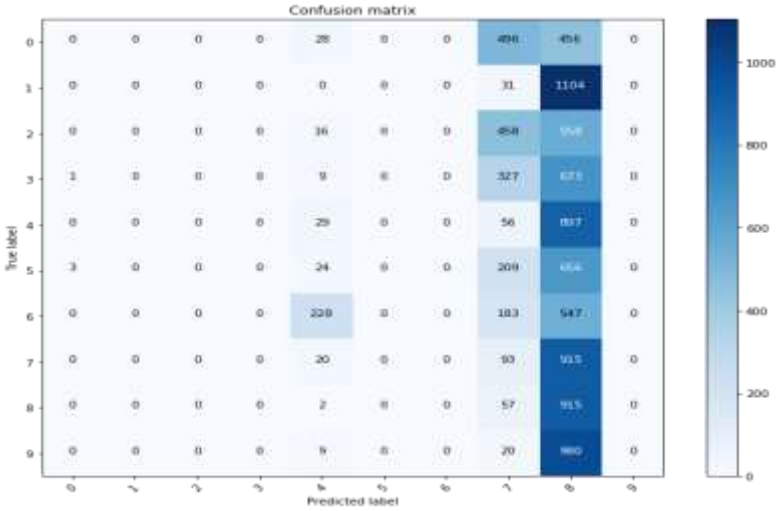
Other Observations:

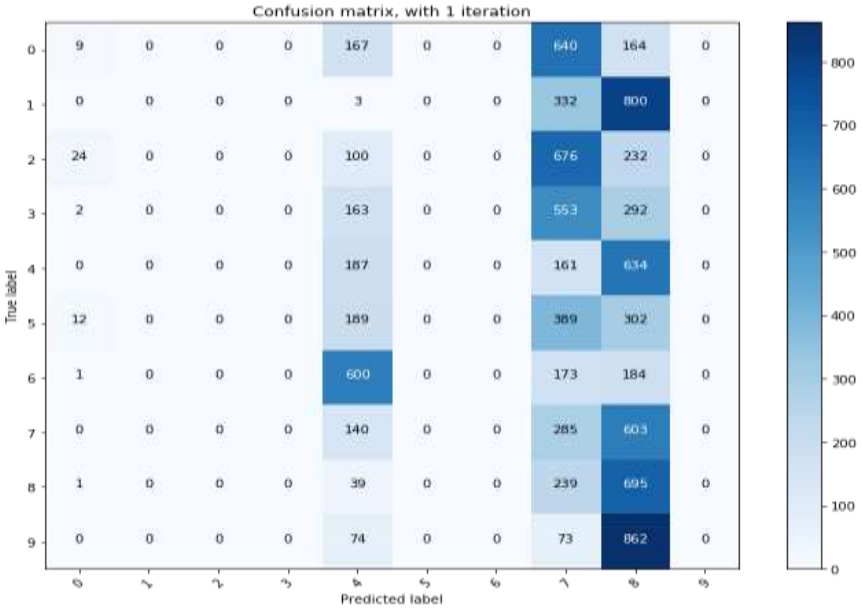
a) *We can see that as the number of hidden layers increase the accuracy decreases because the architecture becomes complex*
b) *Also, the observation is that deepNN performs best with less number of hidden layers*
c) *In terms of accuracy neural network is slightly better than the deepNN*
d) *As the number of hidden layers increases the execution time increases*

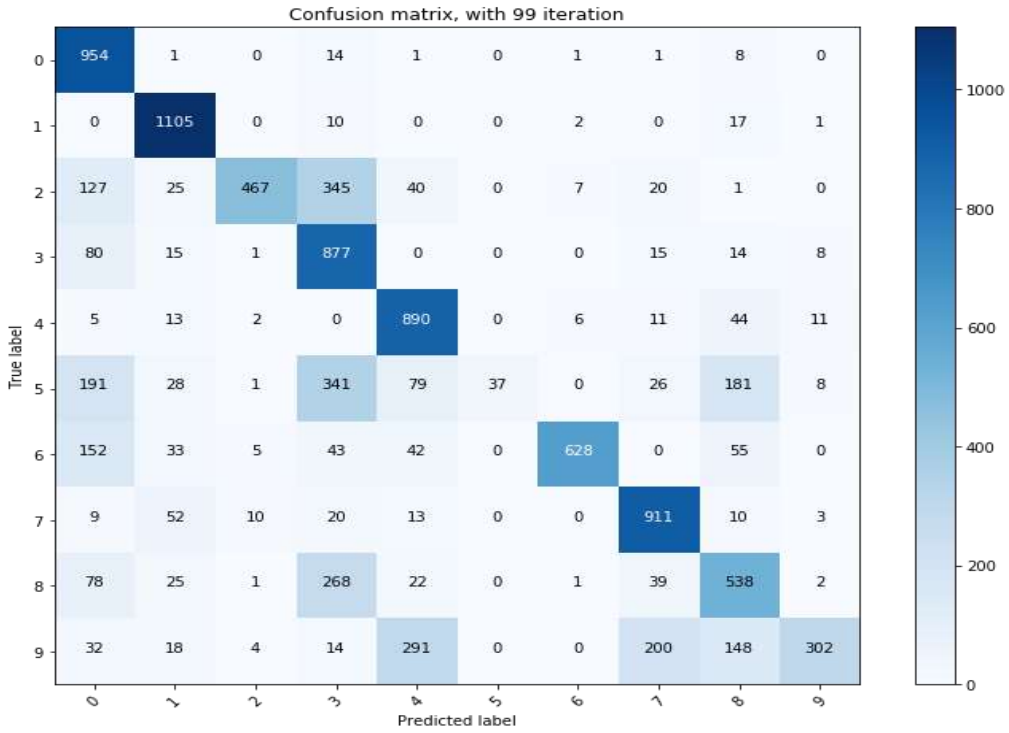
3. *Convolutional Neural Network*

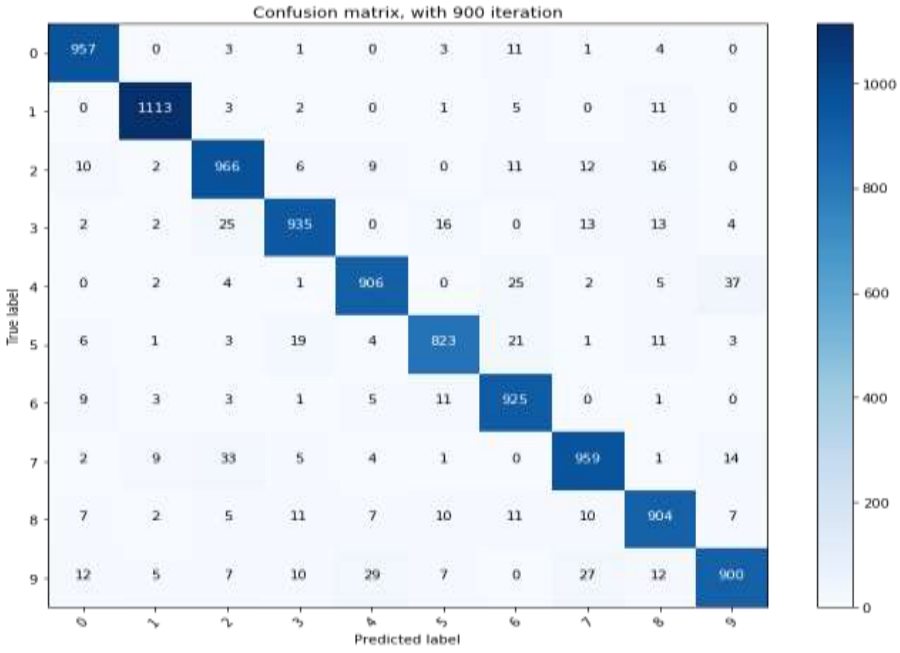
We have plotted the accuracy for the convolutional neural network by plotting confusion matrix for different number of iterations

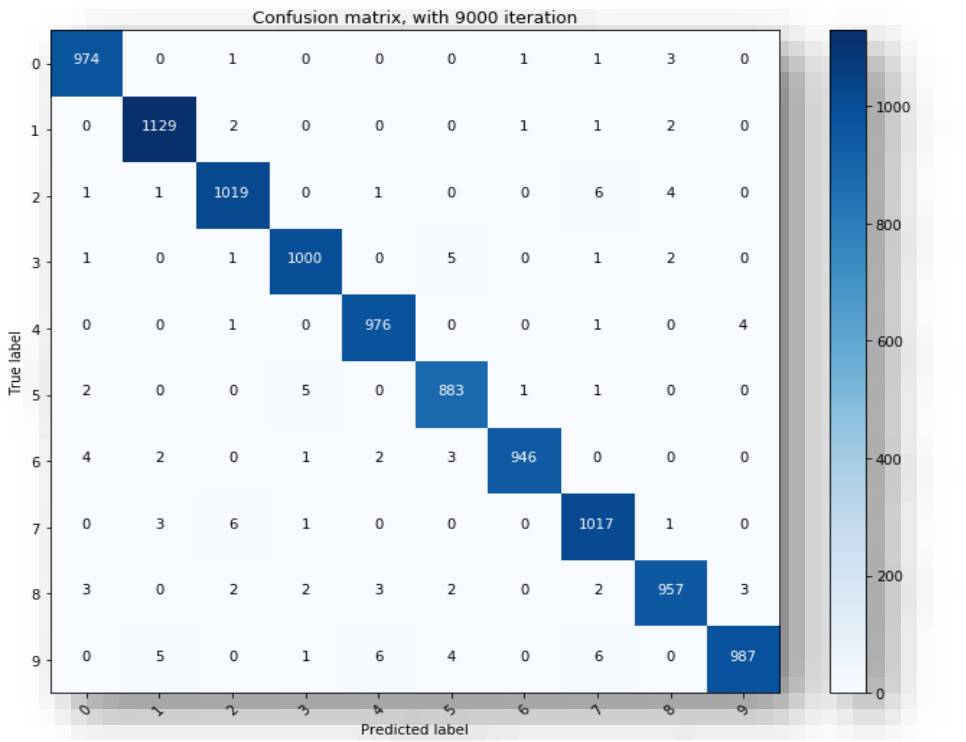
Confusion Matrix






Confusion matrix, with 99 iteration
True label
Predicted label
954 1 0 14 1 0 1 1 8 0
0 1105 0 10 0 0 2 0 17 1
127 25 467 345 40 0 7 20 1 0
80 15 1 877 0 0 0 15 14 8
5 13 2 0 890 0 6 11 44 11
191 28 1 341 79 37 0 26 181 8
152 33 5 43 42 0 628 0 55 0
9 52 10 20 13 0 0 911 10 3
78 25 1 268 22 0 1 39 538 2
32 18 4 14 291 0 0 200 148 302
0 1 2 3 4 5 6 7 8 9
1000 800 600 400 200 0


Confusion matrix, with 900 iteration
True label
Predicted label
957 0 3 1 0 3 11 1 4 0
0 1113 3 2 0 1 5 0 11 0
10 2 966 6 9 0 11 12 16 0
2 2 25 935 0 16 0 13 13 4
0 2 4 1 906 0 25 2 5 37
6 1 3 19 4 823 21 1 11 3
9 3 3 1 5 11 925 0 1 0
2 9 33 5 4 1 0 959 1 14
7 2 5 11 7 10 11 10 904 7
12 5 7 10 29 7 0 27 12 900
0 1 2 3 4 5 6 7 8 9
1000 800 600 400 200 0



Iterations	Time usage	Test Accuracy
Initial Iteration	**0:00:00**	**9.7%**
1	**0:00:00**	**9.8%**
90	**0:00:06**	**61.4%**
900	**0:00:53**	**93.0%**
9000	**0:08:56**	**98.6%**

Based on our observations we can see that as the number of iterations increase the accuracy of our test data set increases however the time elapsed also increases, and our model can predict the right values.
From the confusion matrix it is visible that the number of errors decreases as we increase the number of the iterations

Conclusion

Based on all these results the values that we have selected as our ideal hyper parameters are:

Number of hidden nodes : 100

Regularization Parameter : 5

Number of iterations : 100