ECE 763: Computer Vision - Project 2

Suraj Maniyar
spmaniya@ncsu.edu

Data-set:

The data-set used for this project is the celebA data-set which is a large scale face attributes data-set with more than 200K celebrity images. The images contain large pose variations and background clutter. (http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html)

The face images are taken as it is without cropping so that the model becomes robust to real world situations where a face image may contain background noise. The images are taken in RGB format and re-sized to 60x60 to maintain uniformity.
For non-face images, a random 60x60 block is cropped from the face images from regions which does not contain any face attributes.
A total of 10,000 images are extracted for face and non-face respectively. Also 1000 images of face and non-face are taken for the purpose of testing the model.
The project is implemented on Intel i3 (5th Generation) quad-core processor with a 4 GB RAM. Because of this hardware constraint and to increase efficiency and reduce computation time, 6000 images of face and non-face each are considered for training and 600 images of face and non-face each are taken for testing.

Methodology

A Convolutional Neural Network (CNN) is implemented to train on the images.

- The architecture of the CNN implemented is :

1) Input images dimension - 60x60x3
2) Convolutional Layer with 16 filters of size 5x5 each
3) ReLu activation
4) Max pooling of size (2x2) and strides = 1
5) Convolutional Layer with 16 filters of size 5x5 each
6) ReLu activation
7) Max pooling of size (2x2) and strides = 1
8) Flattened Fully Connected Layer with 3600 (15x15x16) neurons
9) ReLu activation
10) Fully Connected Layer with 128 neurons
11) Fully Connected Layer with 2 Neurons (Output)
12) Softmax activation

The batch-size of the data is kept as 64.
The total number of epochs for training is 10

- The optimizer used is the Adam Optimizer.

- Data Pre-processing:-

The training data is standardized to lie between the range of 0 to 1

Baby sitting procedure:-

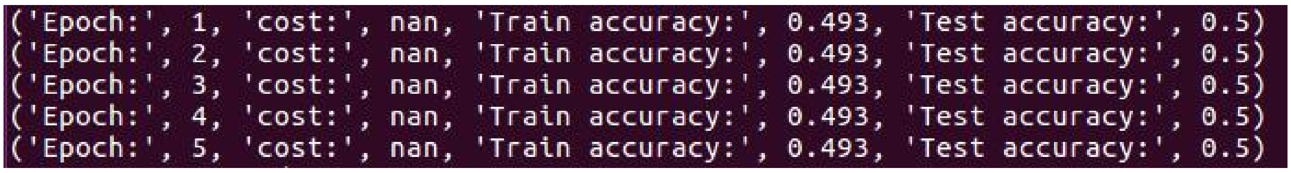
The hyper-parameters which we try to tune are the learning rate and the regularization constant of the Network. The process of choosing the optimum parameters is known as baby sitting the learning process.

The drop out probability is kept constant and equal to 0.2. We only tune the learning rate and the regularization factor.

Learning Rate :-

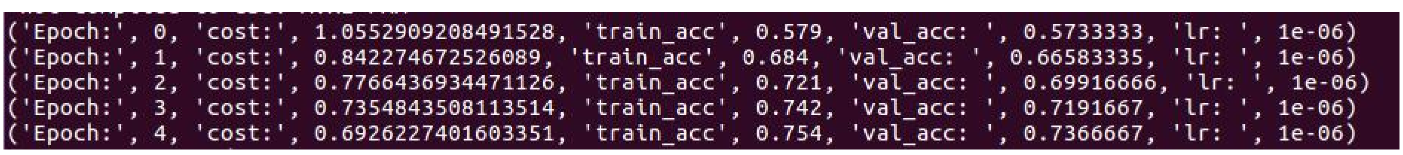
1) First we keep the regularization constant as 0, and vary the learning rate to see the performance. The following are the results:-

With Regularization = 0
Learning Rate = 1e6



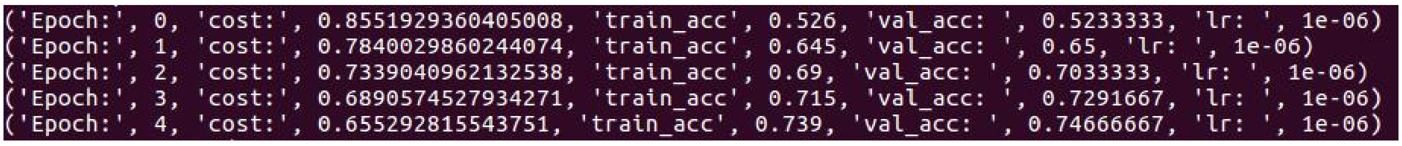
```
('Epoch:', 1, 'cost:', nan, 'Train accuracy:', 0.493, 'Test accuracy:', 0.5)
('Epoch:', 2, 'cost:', nan, 'Train accuracy:', 0.493, 'Test accuracy:', 0.5)
('Epoch:', 3, 'cost:', nan, 'Train accuracy:', 0.493, 'Test accuracy:', 0.5)
('Epoch:', 4, 'cost:', nan, 'Train accuracy:', 0.493, 'Test accuracy:', 0.5)
('Epoch:', 5, 'cost:', nan, 'Train accuracy:', 0.493, 'Test accuracy:', 0.5)
```

With Regularization = 0
Learning Rate = 1e-6



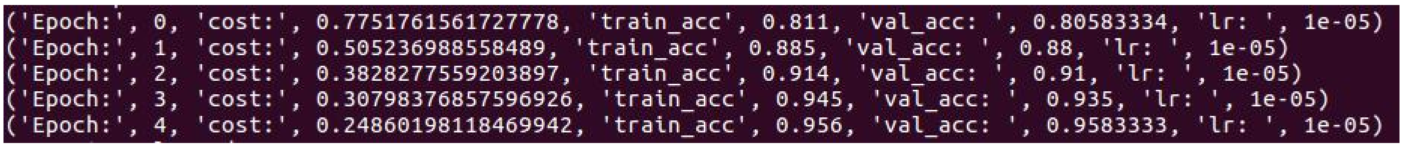
```
('Epoch:', 0, 'cost:', 1.0552909208491528, 'train_acc', 0.579, 'val_acc: ', 0.5733333, 'lr: ', 1e-06)
('Epoch:', 1, 'cost:', 0.842274672526089, 'train_acc', 0.684, 'val_acc: ', 0.66583335, 'lr: ', 1e-06)
('Epoch:', 2, 'cost:', 0.7766436934471126, 'train_acc', 0.721, 'val_acc: ', 0.69916666, 'lr: ', 1e-06)
('Epoch:', 3, 'cost:', 0.7354843508113514, 'train_acc', 0.742, 'val_acc: ', 0.7191667, 'lr: ', 1e-06)
('Epoch:', 4, 'cost:', 0.6926227401603351, 'train_acc', 0.754, 'val_acc: ', 0.7366667, 'lr: ', 1e-06)
```

With Regularization = 0.000001
Learning Rate = 1e-6



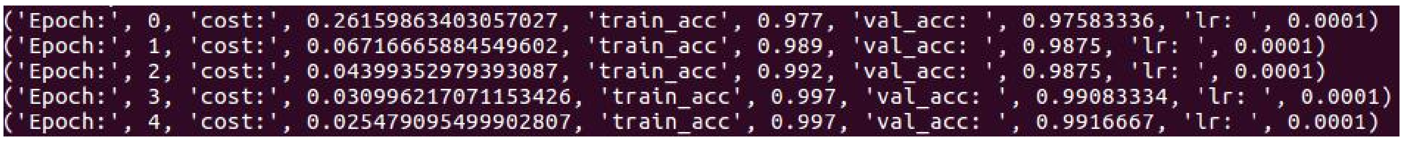
```
('Epoch:', 0, 'cost:', 0.8551929360405008, 'train_acc', 0.526, 'val_acc: ', 0.5233333, 'lr: ', 1e-06)
('Epoch:', 1, 'cost:', 0.7840029860244074, 'train_acc', 0.645, 'val_acc: ', 0.65, 'lr: ', 1e-06)
('Epoch:', 2, 'cost:', 0.7339040962132538, 'train_acc', 0.69, 'val_acc: ', 0.7033333, 'lr: ', 1e-06)
('Epoch:', 3, 'cost:', 0.6890574527934271, 'train_acc', 0.715, 'val_acc: ', 0.7291667, 'lr: ', 1e-06)
('Epoch:', 4, 'cost:', 0.655292815543751, 'train_acc', 0.739, 'val_acc: ', 0.74666667, 'lr: ', 1e-06)
```

With Regularization = 0.000001
Learning Rate = 1e-5



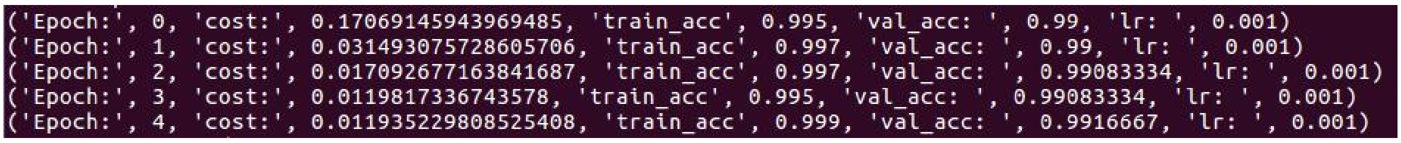
```
('Epoch:', 0, 'cost:', 0.7751761561727778, 'train_acc', 0.811, 'val_acc: ', 0.80583334, 'lr: ', 1e-05)
('Epoch:', 1, 'cost:', 0.505236988558489, 'train_acc', 0.885, 'val_acc: ', 0.88, 'lr: ', 1e-05)
('Epoch:', 2, 'cost:', 0.3828277559203897, 'train_acc', 0.914, 'val_acc: ', 0.91, 'lr: ', 1e-05)
('Epoch:', 3, 'cost:', 0.30798376857596926, 'train_acc', 0.945, 'val_acc: ', 0.935, 'lr: ', 1e-05)
('Epoch:', 4, 'cost:', 0.24860198118469942, 'train_acc', 0.956, 'val_acc: ', 0.9583333, 'lr: ', 1e-05)
```

With Regularization = 0.000001
Learning Rate = 1e-4



```
('Epoch:', 0, 'cost:', 0.26159863403057027, 'train_acc', 0.977, 'val_acc: ', 0.97583336, 'lr: ', 0.0001)
('Epoch:', 1, 'cost:', 0.06716665884549602, 'train_acc', 0.989, 'val_acc: ', 0.9875, 'lr: ', 0.0001)
('Epoch:', 2, 'cost:', 0.04399352979393087, 'train_acc', 0.992, 'val_acc: ', 0.9875, 'lr: ', 0.0001)
('Epoch:', 3, 'cost:', 0.030996217071153426, 'train_acc', 0.997, 'val_acc: ', 0.99083334, 'lr: ', 0.0001)
('Epoch:', 4, 'cost:', 0.025479095499902807, 'train_acc', 0.997, 'val_acc: ', 0.9916667, 'lr: ', 0.0001)
```

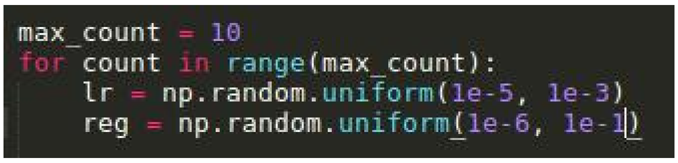
With Regularization = 0.000001
Learning Rate = 1e-3



```
('Epoch:', 0, 'cost:', 0.17069145943969485, 'train_acc', 0.995, 'val_acc: ', 0.99, 'lr: ', 0.001)
('Epoch:', 1, 'cost:', 0.031493075728605706, 'train_acc', 0.997, 'val_acc: ', 0.99, 'lr: ', 0.001)
('Epoch:', 2, 'cost:', 0.017092677163841687, 'train_acc', 0.997, 'val_acc: ', 0.99083334, 'lr: ', 0.001)
('Epoch:', 3, 'cost:', 0.0119817336743578, 'train_acc', 0.995, 'val_acc: ', 0.99083334, 'lr: ', 0.001)
('Epoch:', 4, 'cost:', 0.011935229808525408, 'train_acc', 0.999, 'val_acc: ', 0.9916667, 'lr: ', 0.001)
```

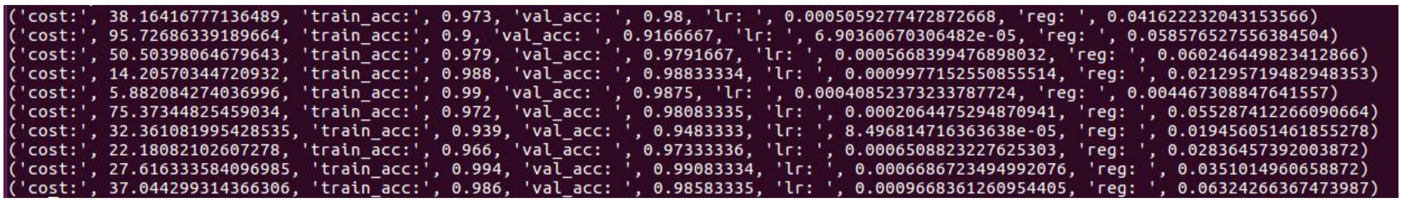
Form the previous results, we observe that after just 5 epochs the results (Train accuracy) start to converge.
A learning rate of 1e6 gives NaN as the cost which means the the learning rate is too high.
Similarly, with a learning rate of 1e-6   the cost hardly changes so it is not a good choice to begin with.
On the other hand, a learning rate of 1e-5 gives 95% accuracy after just 5 epochs. Similarly, learning rate of 1e-3 gives very good results in 5 epochs while the results with 1e-4 are intermediate

Grid Search
So we choose the interval of {1e-5, 1e-3} as the optimum range for learning rate to perform a grid search.
We randomly choose the interval {1e-6, 1e-1} as the interval for the regularization parameter.
So the code snippet for the grid search looks like:-



```
max_count = 10
for count in range(max_count):
    lr = np.random.uniform(1e-5, 1e-3)
    reg = np.random.uniform(1e-6, 1e-1)
```

We train the network for only single epoch during grid search.
The results of grid search in the given interval are:-

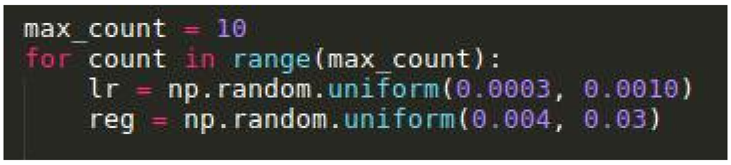


```
('cost:', 38.16416777136489, 'train_acc:', 0.973, 'val_acc: ', 0.98, 'lr: ', 0.0005059277472872668, 'reg: ', 0.041622232043153566)
('cost:', 95.72686339189664, 'train_acc:', 0.9, 'val_acc: ', 0.9166667, 'lr: ', 6.90360670306482e-05, 'reg: ', 0.058576527556384504)
('cost:', 50.50398064679643, 'train_acc:', 0.979, 'val_acc: ', 0.9791667, 'lr: ', 0.0005668399476898032, 'reg: ', 0.060246449823412866)
('cost:', 14.20570344720932, 'train_acc:', 0.988, 'val_acc: ', 0.98833334, 'lr: ', 0.0009977152550855514, 'reg: ', 0.021295719482948353)
('cost:', 5.882084274036996, 'train_acc:', 0.99, 'val_acc: ', 0.9875, 'lr: ', 0.00040852373233787724, 'reg: ', 0.004467308847641557)
('cost:', 75.37344825459034, 'train_acc:', 0.972, 'val_acc: ', 0.98083335, 'lr: ', 0.0002064475294870941, 'reg: ', 0.055287412266090664)
('cost:', 32.361081995428535, 'train_acc:', 0.939, 'val_acc: ', 0.9483333, 'lr: ', 8.496814716363638e-05, 'reg: ', 0.019456051461855278)
('cost:', 22.18082102607278, 'train_acc:', 0.966, 'val_acc: ', 0.97333336, 'lr: ', 0.0006508823227625303, 'reg: ', 0.02836457392003872)
('cost:', 27.616333584096985, 'train_acc:', 0.994, 'val_acc: ', 0.99083334, 'lr: ', 0.000668686723494992076, 'reg: ', 0.0351014960658872)
('cost:', 37.044299314366306, 'train_acc:', 0.986, 'val_acc: ', 0.98583335, 'lr: ', 0.0009668361260954405, 'reg: ', 0.06324266367473987)
```

From the above results, we can see that the cost is lower and the train accuracy is maximum when the values of the parameter are around:-
Learning Rate -> {0.000408, 0.000997, 0.000650}
Regularization Parameter -> {0.004467, 0.021295, 0.028364}

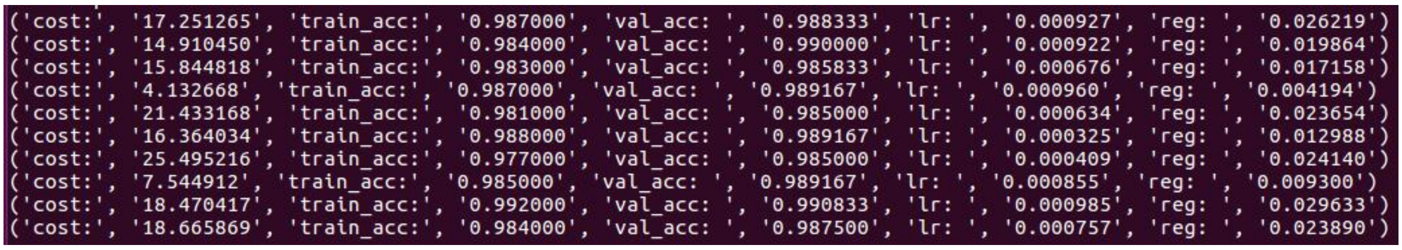
So to fine tune the parameters we choose the next interval as:-

Learning Rate - {0.000300, 0.001000}
Regularization Parameter - {0.004000, 0.030000}



```
max_count = 10
for count in range(max_count):
    lr = np.random.uniform(0.0003, 0.0010)
    reg = np.random.uniform(0.004, 0.03)
```

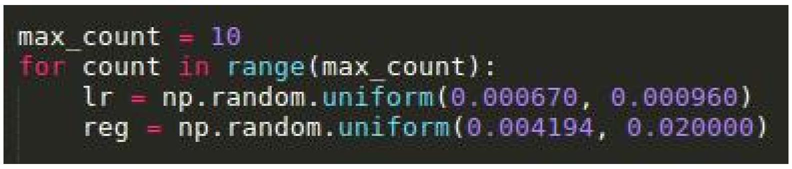
The results of the grid search are:-



```
('cost:', '17.251265', 'train_acc:', '0.987000', 'val_acc: ', '0.988333', 'lr: ', '0.000927', 'reg: ', '0.026219')
('cost:', '14.910450', 'train_acc:', '0.984000', 'val_acc: ', '0.990000', 'lr: ', '0.000922', 'reg: ', '0.019864')
('cost:', '15.844818', 'train_acc:', '0.983000', 'val_acc: ', '0.985833', 'lr: ', '0.000676', 'reg: ', '0.017158')
('cost:', '4.132668', 'train_acc:', '0.987000', 'val_acc: ', '0.989167', 'lr: ', '0.000960', 'reg: ', '0.004194')
('cost:', '21.433168', 'train_acc:', '0.981000', 'val_acc: ', '0.985000', 'lr: ', '0.000634', 'reg: ', '0.023654')
('cost:', '16.364034', 'train_acc:', '0.988000', 'val_acc: ', '0.989167', 'lr: ', '0.000325', 'reg: ', '0.012988')
('cost:', '25.495216', 'train_acc:', '0.977000', 'val_acc: ', '0.985000', 'lr: ', '0.000409', 'reg: ', '0.024140')
('cost:', '7.544912', 'train_acc:', '0.985000', 'val_acc: ', '0.989167', 'lr: ', '0.000855', 'reg: ', '0.009300')
('cost:', '18.470417', 'train_acc:', '0.992000', 'val_acc: ', '0.990833', 'lr: ', '0.000985', 'reg: ', '0.029633')
('cost:', '18.665869', 'train_acc:', '0.984000', 'val_acc: ', '0.987500', 'lr: ', '0.000757', 'reg: ', '0.023890')
```

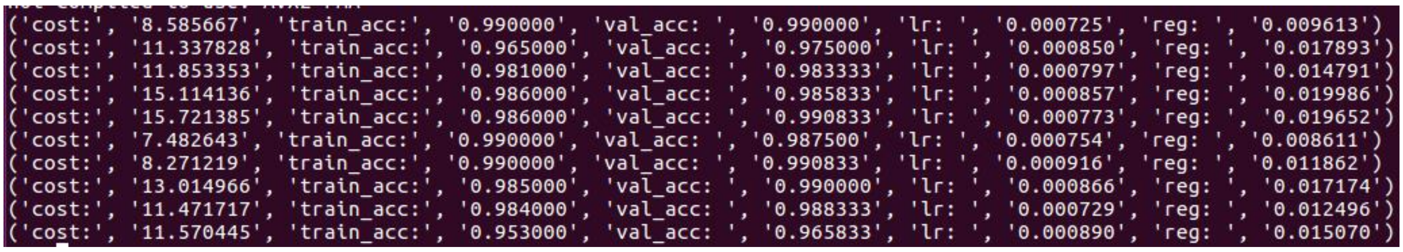
From the above results, we can see that the cost is lower and the train accuracy is maximum when the values of the parameter are around:-
Learning Rate -> {0.000960, 0.000855, 0.000922, 0.000676}
Regularization Parameter -> {0.004194, 0.009300, 0.019864, 0.017158}
So to fine tune the parameters we choose the next interval as:-

Learning Rate - {0.000670, 0.000960}
Regularization Parameter - {0.004194, 0.020000}



```python
max_count = 10
for count in range(max_count):
    lr = np.random.uniform(0.000670, 0.000960)
    reg = np.random.uniform(0.004194, 0.020000)
```

The results of the grid search are:-



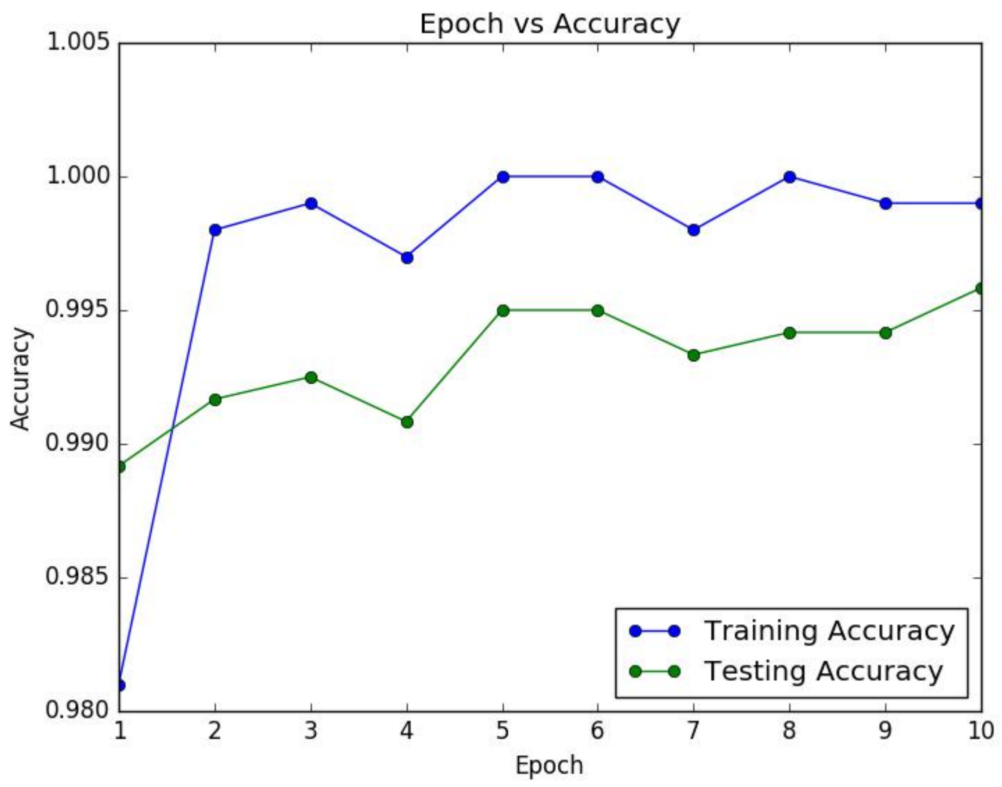
```
('cost:', '8.585667', 'train_acc:', '0.990000', 'val_acc: ', '0.990000', 'lr: ', '0.000725', 'reg: ', '0.009613')
('cost:', '11.337828', 'train_acc:', '0.965000', 'val_acc: ', '0.975000', 'lr: ', '0.000850', 'reg: ', '0.017893')
('cost:', '11.853353', 'train_acc:', '0.981000', 'val_acc: ', '0.983333', 'lr: ', '0.000797', 'reg: ', '0.014791')
('cost:', '15.114136', 'train_acc:', '0.986000', 'val_acc: ', '0.985833', 'lr: ', '0.000857', 'reg: ', '0.019986')
('cost:', '15.721385', 'train_acc:', '0.986000', 'val_acc: ', '0.990833', 'lr: ', '0.000773', 'reg: ', '0.019652')
('cost:', '7.482643', 'train_acc:', '0.990000', 'val_acc: ', '0.987500', 'lr: ', '0.000754', 'reg: ', '0.008611')
('cost:', '8.271219', 'train_acc:', '0.990000', 'val_acc: ', '0.990833', 'lr: ', '0.000916', 'reg: ', '0.011862')
('cost:', '13.014966', 'train_acc:', '0.985000', 'val_acc: ', '0.990000', 'lr: ', '0.000866', 'reg: ', '0.017174')
('cost:', '11.471717', 'train_acc:', '0.984000', 'val_acc: ', '0.988333', 'lr: ', '0.000729', 'reg: ', '0.012496')
('cost:', '11.570445', 'train_acc:', '0.953000', 'val_acc: ', '0.965833', 'lr: ', '0.000890', 'reg: ', '0.015070')
```

Again, the minimum cost and maximum accuracy occurs at the following places:-
Learning Rate -> {0.000754, 0.000916, 0.000725}
Regularization Parameter -> {0.008611, 0.011862, 0.009613}

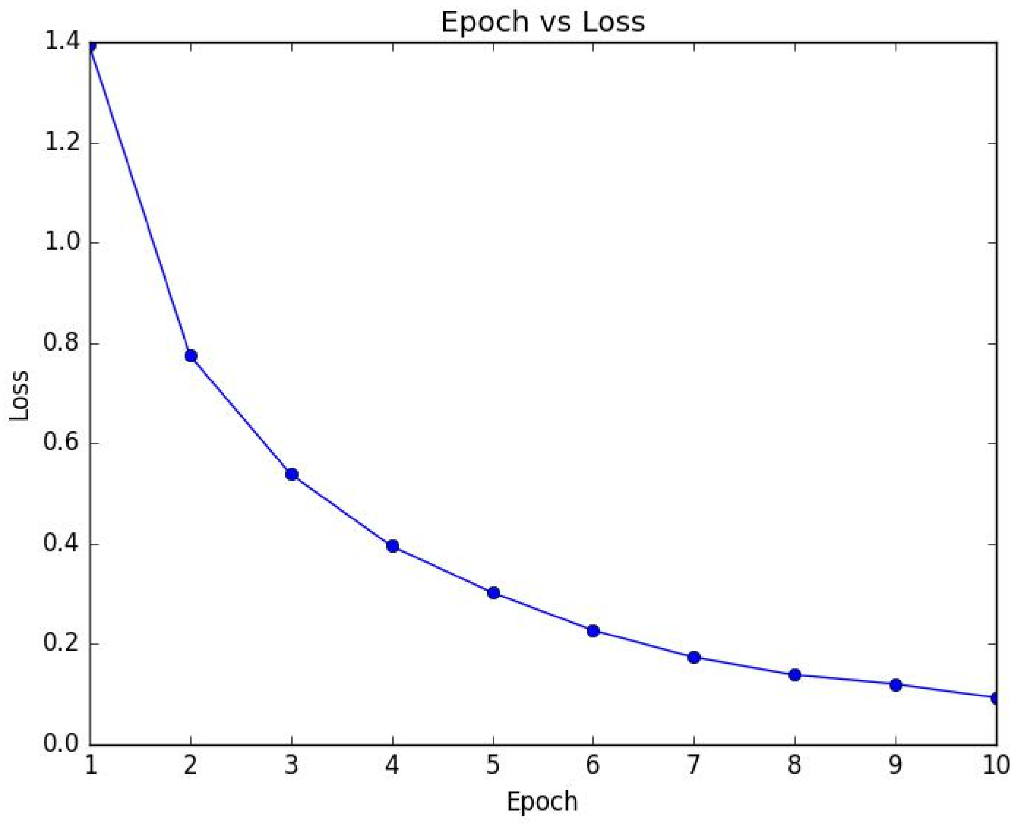
So on an average the ideal parameters for the network would be:-

Learning Rate = 0.0007983
Regularization Parameter = 0.0100286

Now we train the network on the entire data with these hyper-parameters.
The training and the testing data is standardized (lies between 0 to 1) for pre-processing.
The results on pre-processed data are:-

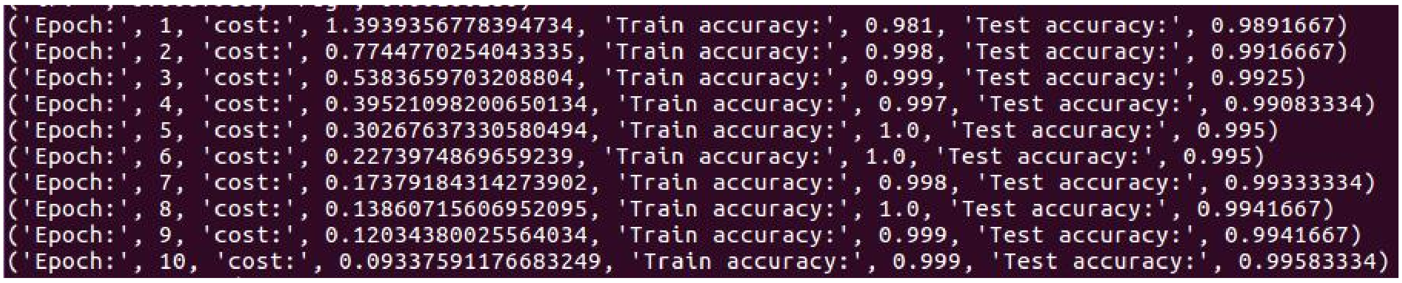


Epoch vs Accuracy
Accuracy
Epoch
1.005
1.000
0.995
0.990
0.985
0.980
1
2
3
4
5
6
7
8
9
10
Training Accuracy
Testing Accuracy



Epoch vs Loss
Loss
Epoch
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1
2
3
4
5
6
7
8
9
10

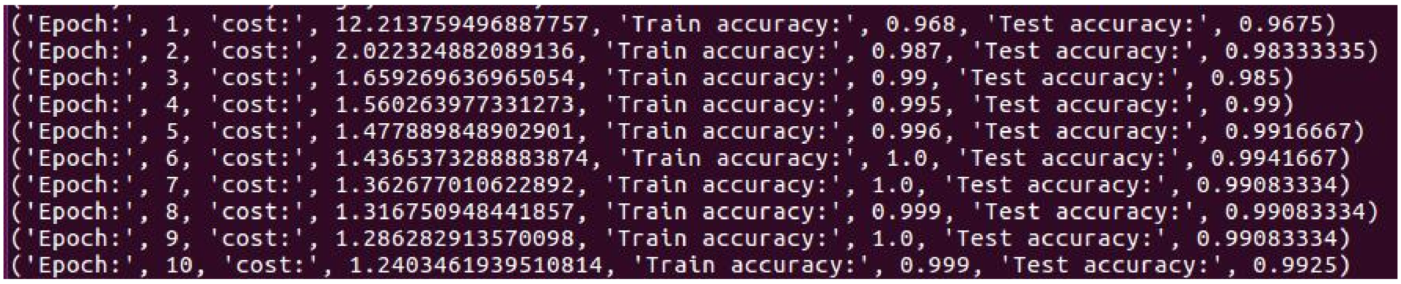
The following are the results after every epoch:-



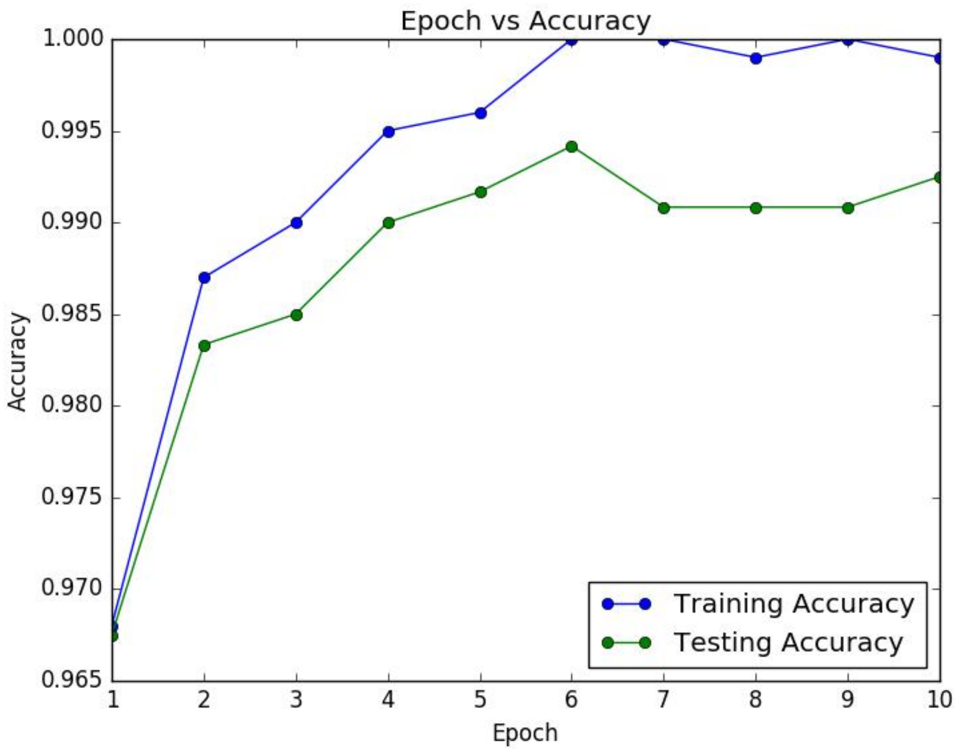
```
('Epoch:', 1, 'cost:', 1.3939356778394734, 'Train accuracy:', 0.981, 'Test accuracy:', 0.9891667)
('Epoch:', 2, 'cost:', 0.7744770254043335, 'Train accuracy:', 0.998, 'Test accuracy:', 0.9916667)
('Epoch:', 3, 'cost:', 0.5383659703208804, 'Train accuracy:', 0.999, 'Test accuracy:', 0.9925)
('Epoch:', 4, 'cost:', 0.39521098200650134, 'Train accuracy:', 0.997, 'Test accuracy:', 0.99083334)
('Epoch:', 5, 'cost:', 0.30267637330580494, 'Train accuracy:', 1.0, 'Test accuracy:', 0.995)
('Epoch:', 6, 'cost:', 0.2273974869659239, 'Train accuracy:', 1.0, 'Test accuracy:', 0.995)
('Epoch:', 7, 'cost:', 0.17379184314273902, 'Train accuracy:', 0.998, 'Test accuracy:', 0.99333334)
('Epoch:', 8, 'cost:', 0.13860715606952095, 'Train accuracy:', 1.0, 'Test accuracy:', 0.9941667)
('Epoch:', 9, 'cost:', 0.12034380025564034, 'Train accuracy:', 0.999, 'Test accuracy:', 0.9941667)
('Epoch:', 10, 'cost:', 0.09337591176683249, 'Train accuracy:', 0.999, 'Test accuracy:', 0.99583334)
```

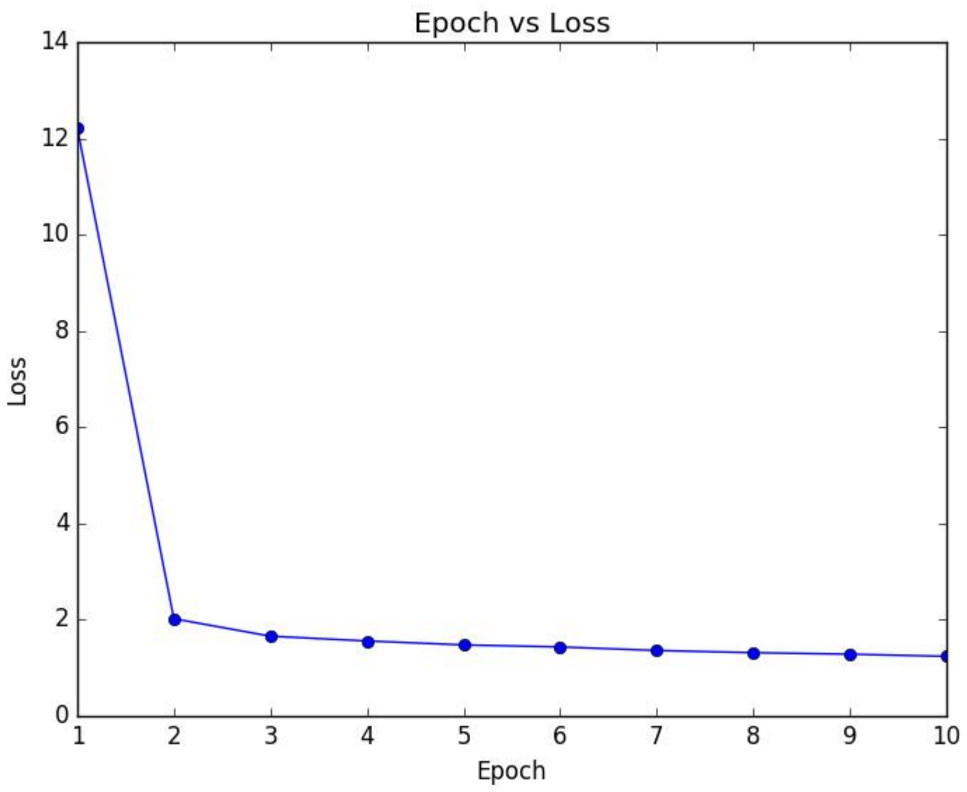
The final accuracy of the model after 10 epochs is **99.58%**
We can also note that the initial cost is very low (approx. 1.393935) and keeps on reducing after every epoch.

If the same model is run on raw data (Not standardized), the results obtained are:-



```
('Epoch:', 1, 'cost:', 12.213759496887757, 'Train accuracy:', 0.968, 'Test accuracy:', 0.9675)
('Epoch:', 2, 'cost:', 2.022324882089136, 'Train accuracy:', 0.987, 'Test accuracy:', 0.98333335)
('Epoch:', 3, 'cost:', 1.659269636965054, 'Train accuracy:', 0.99, 'Test accuracy:', 0.985)
('Epoch:', 4, 'cost:', 1.560263977331273, 'Train accuracy:', 0.995, 'Test accuracy:', 0.99)
('Epoch:', 5, 'cost:', 1.477889848902901, 'Train accuracy:', 0.996, 'Test accuracy:', 0.9916667)
('Epoch:', 6, 'cost:', 1.4365373288883874, 'Train accuracy:', 1.0, 'Test accuracy:', 0.9941667)
('Epoch:', 7, 'cost:', 1.362677010622892, 'Train accuracy:', 1.0, 'Test accuracy:', 0.99083334)
('Epoch:', 8, 'cost:', 1.316750948441857, 'Train accuracy:', 0.999, 'Test accuracy:', 0.99083334)
('Epoch:', 9, 'cost:', 1.286282913570098, 'Train accuracy:', 1.0, 'Test accuracy:', 0.99083334)
('Epoch:', 10, 'cost:', 1.2403461939510814, 'Train accuracy:', 0.999, 'Test accuracy:', 0.9925)
```





The final accuracy of the model after 10 epochs is **99.25%**
We can note that the cost of in this iteration is very high (approx 12 at the start). This is because the model parameters have not been optimized to run on raw data.
