Project Assignment 2
Project Report

First Name: Viswa Nihar
Last Name: Nukala
UBID: 50414392

First Name: Nithin Sastry
Last Name: Tellapuri
UBID: 50365532

First Name: Amartya
Last Name: Banerjee
UBID: 50425019

Introduction:

In this assignment, we must implement a Multilayer Perceptron and evaluate its performance in classifying handwritten digits. Same code must be used to analyze a CelebA dataset. We should also compare the performance with a deep neural network and Convolutional Neural Network, which are implemented using tensor flow.

Pre-Processing:

The first thing which needs to be done to the dataset is to do pre-processing. We do the following things in pre-processing:

- Divide the data set into different data sets to train and the test the Neural Network model we are designing
- Do feature selection

As part of the first point, we divide the dataset train data, test data and validation data which is used to train, test, and validate the model.

As part of the second point, we do feature selection based on the description given in the pdf. It is mentioned that there are many rows which may have the same data, hence this data will not be useful and will just be repetitive and we can ignore this data. After applying the above logic to the dataset, we can see the following are the indices which are distinct and are hence useful:

```
12   13   14   15   32   33   34   35   36   37   38   39   40   41   42   43   44   45   46   47   48   49   50   51   58
59   60   61   62   63   64   65   66   67   68   69   70   71   72   73   74   75   76   77   78   79   80   81   86   87
88   89   90   91   92   93   94   95   96   97   98   99   100  101  102  103  104  105  106  107  108  109  110
113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133
134  135  136  137  138  139  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156
157  158  159  160  161  162  163  164  165  166  167  169  170  171  172  173  174  175  176  177  178
179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199
200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220
221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241
242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262
263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283
284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304
305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325
326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343  344  345  346
347  348  349  350  351  352  353  354  355  356  357  358  359  360  361  362  363  364  365  366  367
368  369  370  371  372  373  374  375  376  377  378  379  380  381  382  383  384  385  386  387  388
389  390  391  392  393  394  395  396  397  398  399  400  401  402  403  404  405  406  407  408  409
410  411  412  413  414  415  416  417  418  419  420  421  422  423  424  425  426  427  428  429  430
431  432  433  434  435  436  437  438  439  440  441  442  443  444  445  446  447  448  449  450  451
452  453  454  455  456  457  458  459  460  461  462  463  464  465  466  467  468  469  470  471  472
473  474  475  477  478  479  480  481  482  483  484  485  486  487  488  489  490  491  492  493  494
495  496  497  498  499  500  501  502  503  504  505  506  507  508  509  510  511  512  513  514  515
516  517  518  519  520  521  522  523  524  525  526  527  528  529  530  531  532  533  534  535  536
537  538  539  540  541  542  543  544  545  546  547  548  549  550  551  552  553  554  555  556  557
558  559  561  562  563  564  565  566  567  568  569  570  571  572  573  574  575  576  577  578  579
580  581  582  583  584  585  586  587  588  589  590  591  592  593  594  595  596  597  598  599  600
601  602  603  604  605  606  607  608  609  610  611  612  613  614  615  616  617  618  619  620  621
622  623  624  625  626  627  628  629  630  631  632  633  634  635  636  637  638  639  640  641  642
643  646  647  648  649  650  651  652  653  654  655  656  657  658  659  660  661  662  663  664  665
666  667  668  669  670  674  675  676  677  678  679  680  681  682  683  684  685  686  687  688  689
690  691  692  693  694  695  696  697  698  702  703  704  705  706  707  708  709  710  711  712  713
714  715  716  717  718  719  720  721  722  723  724  725  726  731  732  733  734  735  736  737  738
739  740  741  742  743  744  745  746  747  748  749  750  751  752  753  760  761  762  763  764  765
766  767  768  769  770  771  772  773  774  775  776  777  778  779
Number of features selected:  717
```

We also need to save all there features in the params.pickle so that these can be used to pre-process the data again for the CelebA dataset.

nnObjFunction:

In this function we implement the logic required for to do both the forward propagation and the backward propagation using regularization term too. To complete this we also use the **sigmoid** function which performs sigmoid over the given value (doesn't matter if it is a scalar

value or matrix or a vector). I have also implemented the **sig_derivative** function which computes the sigmoid derivative value. The return value for this function is an obj_value and an obj_grad which is going to be used by the minimize function.

nnPredict:

This function predicts the values based on the model trained and returns a set of labels. These labels will be used to predict the accuracies for test data, train data and validation data.

Choosing Hyper-Parameters:

Hyper-parameters are ƛ and m. As given in the document we need to choose the best hyper-parameters by testing different values and getting the highest accuracy.

ƛ is the regularization term which is used to prevent over-fitting. We also change the number of hidden units in a layer to change its effect and increase efficiency.

We change the value of ƛ from 10-60 with an increment of 10 and for each ƛ value, we also compute accuracies by changing the value of m from the range of 4-20 with an increment of 4. We also note down the training time taken to train the model to compare it later.

After gathering the data for all the different combinations of we choose the best ƛ and m value with the highest test accuracy.

Following is the top 5 data frame for which the test accuracy was the highest:

λ	m	Train_Accuracy	Validation_Accuracy	Test_Accuracy	Training_Time
30	20	93.492	93.06	93.26	24.107
40	16	93.194	93.11	93.15	23.7246
0	20	93.48	92.88	93.13	25.1082
60	20	92.724	92.61	92.92	23.9794
0	16	93.434	93.13	92.92	24.3978

So, from this table we can see that ƛ=30 and m=20 gives us the highest test accuracy which is **93.26%**

Following is the data with ƛ=30 and varying the m as 4,8,12,16,20:

λ	m	Train_Accuracy	Validation_Accuracy	Test_Accuracy	Training_Time
30	4	63.486	63.53	62.71	15.811
30	8	84.858	84.36	85.01	18.1268
30	12	91.088	90.9	91.14	20.9677
30	16	92.362	92.1	92.01	23.913
30	20	93.492	93.06	93.26	24.107

We also need to plot the values of ƛ (with m=20) vs accuracy which shows the variation of test accuracy, train accuracy and validation accuracy.

Figure (1) shows the above variation discussed. From the figure we can say that initially the data was underfit and as we increase the ƛ value the model continues to underfit until it starts fitting the data properly and hence the accuracy increases. After the optimal point ƛ=30, the

model starts overfitting again. After some point it starts to underfit again.

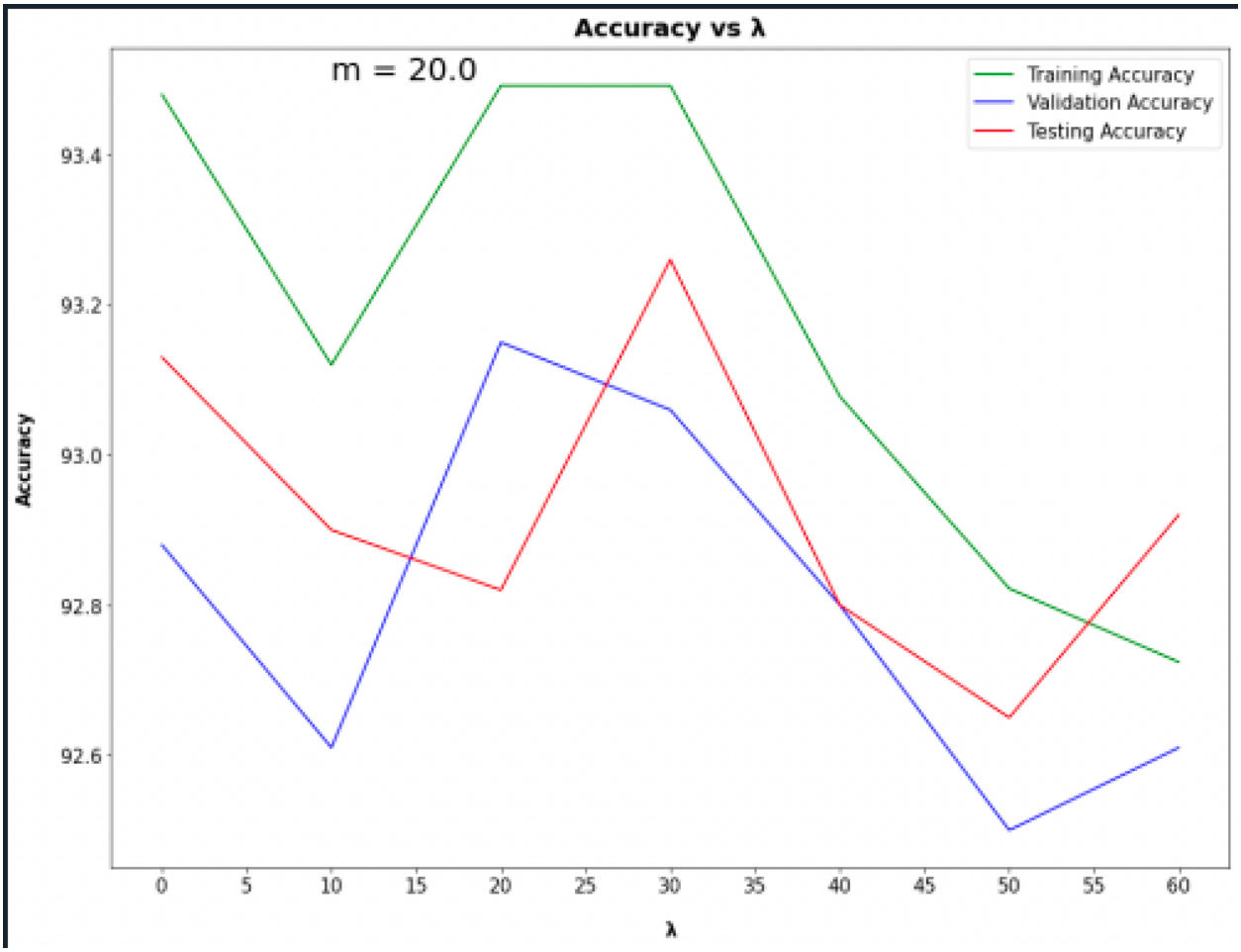

Figure (1)

Figure (2) shows the relationship between the accuracies and the value of hidden nodes (m) by keeping ƛ constant at 30.

We can infer from the graph that as the number of hidden nodes are increased, the accuracy also increases as it reaches the local optimum. We can also see that all the three accuracies almost the same for the fixed ƛ.

We can see that m=20 gives us the best value.

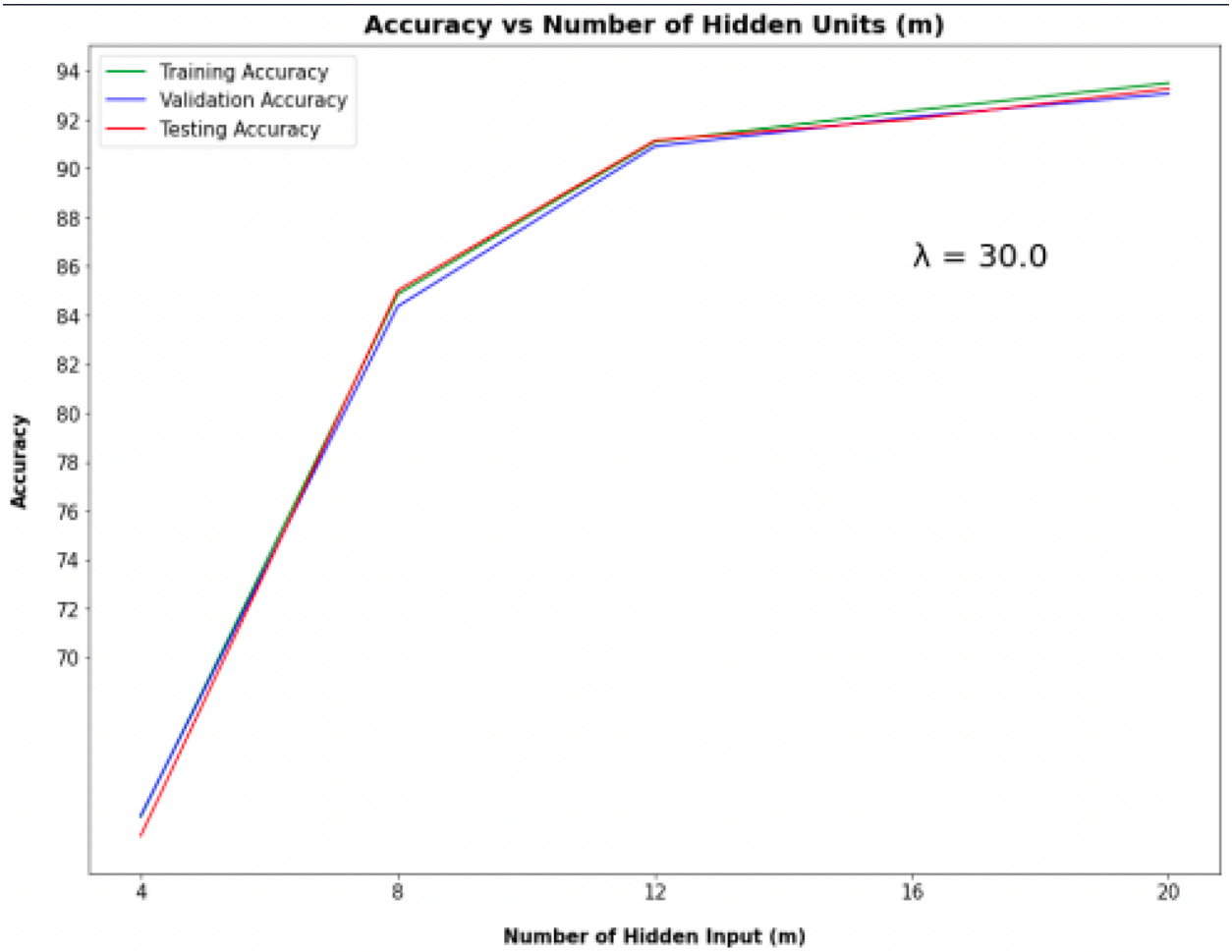

Figure (2)

Figure (3) depicts the training time vs the number of hidden units.

We can infer from the graph that the time take increases as the number of hidden units are increased. This is expected as there are more hidden units, there can be more values which needs to be calculated and that too twice as both forward and backward propagation needs to be done.

Hence, care needs to be taken when choosing hidden units as even though accuracy may increase, the time taken to train increases.

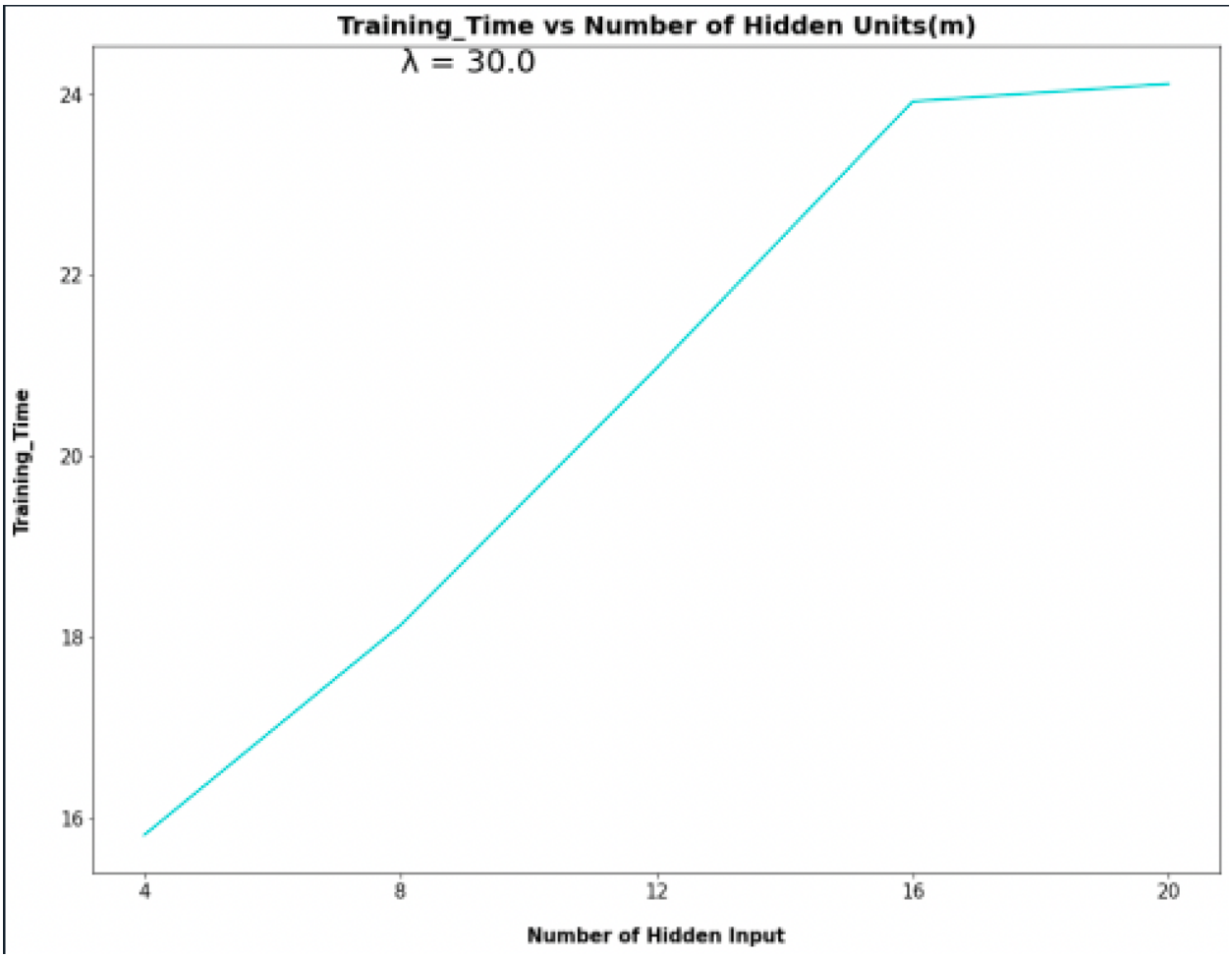

Figure (3)

Accuracy with the best hyper-parameters for handwritten digits:

Following data is observed when the hyper-parameters are
Train Accuracy = 93.482
Validation Accuracy = 93.06
Validation Accuracy = 93.26

Accuracy of classification on CelebA dataset:

I have copied the functions, sigmoid(), nnObjFunction() and nnPredict() and run the facennScript.py with ƛ=10.

Following are the results we get from running facennScript.py:

```
Training set Accuracy:84.70142180094786%

Validation set Accuracy:83.15196998123827%

Test set Accuracy:84.93565480696442%
```

Accuracy from Deep Neural Networks (DNN) using TensorFlow:

Single Layer facennScript.py is run on my local machine and deepnnScript.py with different layers is run on timberlake, hence the large difference in the value of training times.

Number of Layers	Model	Training Time (s)	Accuracy (%)
1	Neural Network	85.402	85.54
3	Deep Neural Network	662.352	78.43
5	Deep Neural Network	750.535	75.78
7	Deep Neural Network	972.507	74.53

We can infer from the table that as we increase the number of layers, the training time also increases. Accuracy also decreases as we increase the number of layers. One of the reasons why this happens is due to overfitting.

Running Convolutional Neural Networks using TensorFlow:

cnnScript.py contains code to run Convolutional Neural Networks using TensorFlow.

We run the CNN by increasing the number of iterations and check how the accuracy and training time varies.

Number of Iterations	Training Time (s)	Accuracy (%)
0	-	9.7
1	0	10.7
100	7	71.0
1000	62	93.6
10000	614	98.7

From the above table, we can say that as we increase the number of iterations, the accuracy also increases. Training also takes a lot of time as we are increasing the number of iterations.

Following is the confusion matrix obtained from the running the CNN for 10000 iterations:

```
Confusion Matrix:
[[ 972    0    0    0    0    1    2    1    4    0]
 [   0 1128    2    0    0    0    0    0    5    0]
 [   0    2 1015    0    1    0    0    3   11    0]
 [   0    0    1 1003    0    1    0    2    3    0]
 [   0    0    0    0  975    0    1    2    3    1]
 [   2    0    0    7    0  872    3    0    6    2]
 [   4    3    0    0    2    3  941    0    5    0]
 [   0    2    7    1    0    0    0 1015    3    0]
 [   3    0    1    1    0    0    1    1  966    1]
 [   1    4    0    3    7    1    1    2    7  983]]
```

Following is the plot of the confusion matrix: