

Submission of the assignment 3 of big data course

Xiaoke(Jimmy) Shen

April 14, 2017

1 Summary of the submission of this assignment

In this submission, single layer, multiple layer neural network and the CNN (Convolutional Neural Networks) are used to do the classify the CIFAR10 dataset provided in the big data course.

Results are shown in figure 1. Details are given in the following sections.

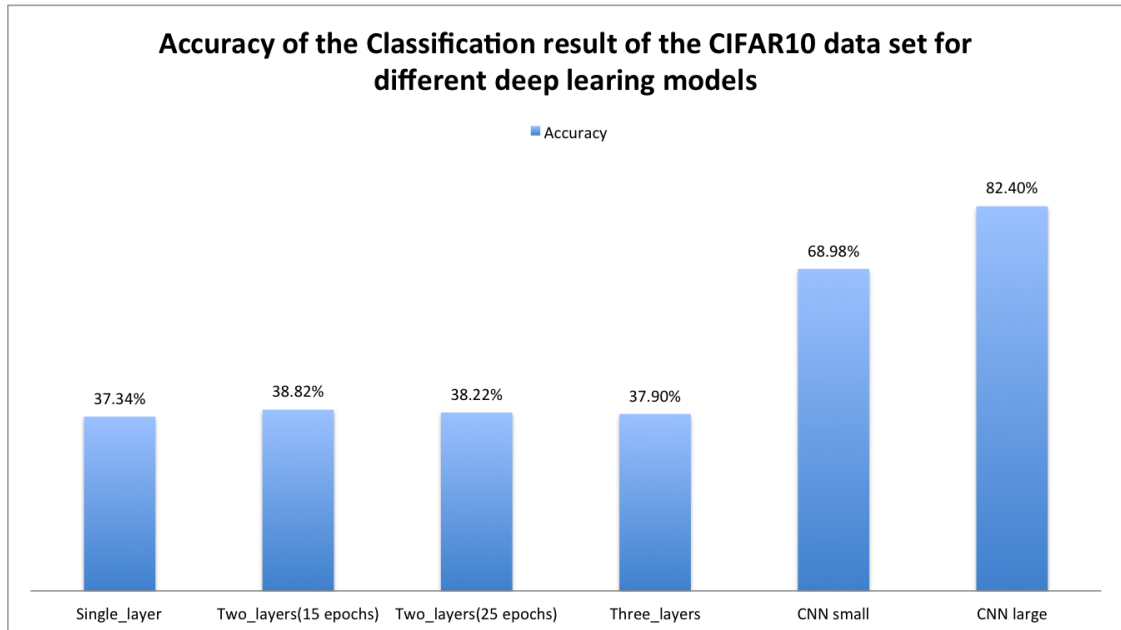


Figure 1: The result of the classification result of the CIFAR10 dataset on different models used in this submission.

2 Part 1

2.1 Description of the assignment part 1

Develop a simple one layer neural network to classify the CIFAR10 dataset. Play with learning rates, number of training steps, training sample size, etc and see how well you can do. This is a harder dataset than MNIST, and the downsampling to greyscale reduces the image information even further, so anything above 50% is acceptable in this step(in the requirement of the updated email, above 30% is acceptable).

2.2 Submission of the part 1

The **one hidden layer** neural network is used here to classify the CIFAR10 dataset. The number of nodes of this hidden layer is the same as the pixel number of each image which is $3 * 32 * 32 = 3072$. The activity function is "relu". The output layer has 10 nodes and the activity function is "softmax". The accuracy of this one layer network is **37.34%** under 10 epochs. The Screenshots of the result is given in figure 2.

The code of this part is provided in the file
big_data_assignment3_single_layer.py.

```
ValueError: Input arrays should have the same number of samples as target arrays. Found 45000
xiaokes-MacBook-Pro:big_data jimmy$ python big_data_assignment3_single_layer.py
Using TensorFlow backend.
('type of dict', <type 'dict'>)
('type of features', <type 'numpy.ndarray'>)
('shape of features', (50000, 3072))
('lbl[0]', array([ 0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.], dtype=float32))
('type of lbl', <type 'numpy.ndarray'>)
('shape of lbl', (50000, 10))
Train on 45000 samples, validate on 5000 samples
Epoch 1/10
104s - loss: 2.0894 - acc: 0.2827 - val_loss: 1.8441 - val_acc: 0.3568
Epoch 2/10
92s - loss: 1.5619 - acc: 0.4850 - val_loss: 1.7895 - val_acc: 0.3660
Epoch 3/10
87s - loss: 1.3259 - acc: 0.5635 - val_loss: 1.8365 - val_acc: 0.3644
Epoch 4/10
87s - loss: 1.1354 - acc: 0.6361 - val_loss: 1.8760 - val_acc: 0.3654
Epoch 5/10
87s - loss: 0.9434 - acc: 0.7121 - val_loss: 1.9326 - val_acc: 0.3666
Epoch 6/10
87s - loss: 0.7482 - acc: 0.7928 - val_loss: 1.9998 - val_acc: 0.3670
Epoch 7/10
87s - loss: 0.5682 - acc: 0.8658 - val_loss: 2.0782 - val_acc: 0.3650
Epoch 8/10
87s - loss: 0.4119 - acc: 0.9230 - val_loss: 2.1457 - val_acc: 0.3646
Epoch 9/10
87s - loss: 0.2884 - acc: 0.9601 - val_loss: 2.2196 - val_acc: 0.3702
Epoch 10/10
87s - loss: 0.1976 - acc: 0.9806 - val_loss: 2.3018 - val_acc: 0.3734
Baseline Accuracy: 37.34%
xiaokes-MacBook-Pro:big_data jimmy$
```

Figure 2: The Screenshots of the result of single layer neural network: input:3072, first hidden layer: $3 * 32 * 32 = 3072$ nodes, output:10 nodes. The loss means the loss of the training data, acc means the accuracy of the training data. The val_loss means the loss of the test data, val_acc means the accuracy of the test data.

3 Part 2

3.1 Description of the assignment part 2

Add a hidden layer to the network. So, instead of going from 3072 inputs to 10 outputs, you'll be going from 3072 inputs to N intermediate neurons, then from N intermediate outputs to 10 outputs. You'll need a second set

of weights and biases, and you'll have to think a bit about what size your weight/bias matrices need to be, but once you've got that right the network itself is just two separate matrix multiplication steps (plus the biases) instead of one. Remember to apply a nonlinearity to the output of the hidden layer; `tf.nn.relu()` is a reasonable choice for this. See how much of an improvement you can get over your single layer network. In addition to learning rate etc. you now have a new hyperparameter to play with: N , the size of your hidden layer. If you're feeling a bit more ambitious, try adding more layers and see what combination of layer numbers and layer sizes works best for you.

3.2 Submission of the part 2

3.2.1 Two hidden layer neural network

The **two hidden layer** neural network is used here to classify the CIFAR10 dataset. The number of nodes of the first hidden layer is the same as the pixel number of each image which is $3 * 32 * 32 = 3072$. The activity function is "relu". For the second hidden layer, the number of nodes is 128 and the activity function is also "relu". The output layer has 10 nodes and the activity function is "softmax".

The accuracy of this one layer network is **38.82%** under 10 epochs (Screenshots of the result is given in figure 3).

The accuracy of this one layer network is **38.22%** under 25 epochs (Screenshots of the result is given in figure 4).

The code of this part is provided in the file `big_data_assignment3_two_layers.py`.

From the result we can get the conclusion below:

- The result of two hidden layer network is slight better than the single layer network.
- When the epoch is bigger enough, there is almost no improvement of the result on the test data
- The accuracy of the training part is approaching to 100% when the number of the epoch increases, however, the result of the accuracy of the test part is far away from 100%. The reason is overfitting. In order to fix the overfitting problem. Dropout is used for this two layers network and the overfitting still exists.

```

xiaokes-MacBook-Pro:big_data jimmy$ python big_data_assignment3_two_layers.py
Using TensorFlow backend.
('type of dict', <type 'dict'>)
('type of features', <type 'numpy.ndarray'>)
('shape of features', (50000, 3072))
('lbl[0]', array([ 0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.], dtype=float32))
('type of lbl', <type 'numpy.ndarray'>)
('shape of lbl', (50000, 10))
Train on 45000 samples, validate on 5000 samples
Epoch 1/10
98s - loss: 2.0824 - acc: 0.2414 - val_loss: 1.8715 - val_acc: 0.3274
Epoch 2/10
104s - loss: 1.5991 - acc: 0.4420 - val_loss: 1.7774 - val_acc: 0.3688
Epoch 3/10
106s - loss: 1.2367 - acc: 0.5785 - val_loss: 1.8043 - val_acc: 0.3844
Epoch 4/10
102s - loss: 0.8294 - acc: 0.7310 - val_loss: 1.9697 - val_acc: 0.3892
Epoch 5/10
100s - loss: 0.4449 - acc: 0.8741 - val_loss: 2.1971 - val_acc: 0.3878
Epoch 6/10
93s - loss: 0.1941 - acc: 0.9592 - val_loss: 2.4371 - val_acc: 0.3926
Epoch 7/10
102s - loss: 0.0825 - acc: 0.9885 - val_loss: 2.7116 - val_acc: 0.3942
Epoch 8/10
99s - loss: 0.0427 - acc: 0.9960 - val_loss: 2.8861 - val_acc: 0.3972
Epoch 9/10
100s - loss: 0.0280 - acc: 0.9974 - val_loss: 3.0222 - val_acc: 0.3908
Epoch 10/10
97s - loss: 0.0250 - acc: 0.9970 - val_loss: 3.1868 - val_acc: 0.3882
Baseline Accuracy: 38.82%

```

Figure 3: The Screenshots of the result of **two hidden layer** neural network: input:3072, first hidden layer: 3072 nodes, second hidden layer: 128 nodes, output:10 nodes. **10 epochs** is used here.

```

xiaokes-MacBook-Pro:big_data jimmy$ python big_data_assignment3_two_layers.py
Using TensorFlow backend.
('type of dict', <type 'dict'>)
('type of features', <type 'numpy.ndarray'>)
('shape of features', (50000, 3072))
('lbl[0]', array([ 0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.], dtype=float32))
('type of lbl', <type 'numpy.ndarray'>)
('shape of lbl', (50000, 10))
Train on 45000 samples, validate on 5000 samples
Epoch 1/25
92s - loss: 2.0824 - acc: 0.2411 - val_loss: 1.8729 - val_acc: 0.3250
Epoch 2/25
92s - loss: 1.6006 - acc: 0.4412 - val_loss: 1.7765 - val_acc: 0.3692
Epoch 3/25
92s - loss: 1.2419 - acc: 0.5773 - val_loss: 1.8003 - val_acc: 0.3822
Epoch 4/25
104s - loss: 0.8430 - acc: 0.7266 - val_loss: 1.9569 - val_acc: 0.3892
Epoch 5/25
109s - loss: 0.4672 - acc: 0.8660 - val_loss: 2.1712 - val_acc: 0.3876
Epoch 6/25
100s - loss: 0.2082 - acc: 0.9535 - val_loss: 2.4319 - val_acc: 0.3902
Epoch 7/25
109s - loss: 0.0888 - acc: 0.9865 - val_loss: 2.6974 - val_acc: 0.3932
Epoch 8/25
92s - loss: 0.0467 - acc: 0.9952 - val_loss: 2.8958 - val_acc: 0.3918
Epoch 9/25
109s - loss: 0.0327 - acc: 0.9968 - val_loss: 3.1033 - val_acc: 0.3854
Epoch 10/25
108s - loss: 0.0332 - acc: 0.9953 - val_loss: 3.1940 - val_acc: 0.3854
Epoch 11/25
97s - loss: 0.0453 - acc: 0.9909 - val_loss: 3.2686 - val_acc: 0.3838
Epoch 12/25
105s - loss: 0.0522 - acc: 0.9894 - val_loss: 3.4023 - val_acc: 0.3752
Epoch 13/25
102s - loss: 0.0391 - acc: 0.9920 - val_loss: 3.4952 - val_acc: 0.3788
Epoch 14/25
100s - loss: 0.0312 - acc: 0.9937 - val_loss: 3.5990 - val_acc: 0.3850
Epoch 15/25
93s - loss: 0.0239 - acc: 0.9950 - val_loss: 3.6524 - val_acc: 0.3848
Epoch 16/25
92s - loss: 0.0203 - acc: 0.9962 - val_loss: 3.7058 - val_acc: 0.3882
Epoch 17/25
92s - loss: 0.0179 - acc: 0.9966 - val_loss: 3.7706 - val_acc: 0.3938
Epoch 18/25
92s - loss: 0.0170 - acc: 0.9968 - val_loss: 3.8560 - val_acc: 0.3808
Epoch 19/25
92s - loss: 0.0174 - acc: 0.9965 - val_loss: 3.8375 - val_acc: 0.3876
Epoch 20/25
92s - loss: 0.0180 - acc: 0.9960 - val_loss: 3.9099 - val_acc: 0.3854
Epoch 21/25
101s - loss: 0.0204 - acc: 0.9954 - val_loss: 3.9247 - val_acc: 0.3804
Epoch 22/25
100s - loss: 0.0221 - acc: 0.9953 - val_loss: 3.9683 - val_acc: 0.3760
Epoch 23/25
109s - loss: 0.0176 - acc: 0.9959 - val_loss: 4.0425 - val_acc: 0.3824
Epoch 24/25
109s - loss: 0.0171 - acc: 0.9961 - val_loss: 4.0807 - val_acc: 0.3786
Epoch 25/25
95s - loss: 0.0163 - acc: 0.9964 - val_loss: 4.1532 - val_acc: 0.3822
Baseline Accuracy: 38.22%

```

Figure 4: The Screenshots of the result of **two hidden layer** neural network: input:3072, first hidden layer: 3072 nodes, second hidden layer: 128 nodes, output:10 nodes. **25 epochs** is used here.

3.2.2 Three hidden layer neural network

The **three hidden layer** neural network is used here to classify the CIFAR10 dataset. The first two hidden layers are the same as the two layer model. The third hidden layer has 64 nodes with an activity function of "relu". The output layer has 10 nodes and the activity function is "softmax". The accuracy of this network is **37.90%**(Screenshots of the result is given in figure 5).

The code of this part is provided in the file `big_data_assignment3_three_layers.py`.

```

xiaokes-MacBook-Pro:big_data jimmy$ python big_data_assignment3_three_layers.py
Using TensorFlow backend.
('type of dict', <type 'dict'>)
('type of features', <type 'numpy.ndarray'>)
('shape of features', (50000, 3072))
('lbl[0]', array([ 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.], dtype=float32))
('type of lbl', <type 'numpy.ndarray'>)
('shape of lbl', (50000, 10))
Train on 45000 samples, validate on 5000 samples
Epoch 1/25
107s - loss: 2.1261 - acc: 0.1929 - val_loss: 1.9541 - val_acc: 0.2506
Epoch 2/25
125s - loss: 1.7402 - acc: 0.3480 - val_loss: 1.8125 - val_acc: 0.3438
Epoch 3/25
111s - loss: 1.3347 - acc: 0.5098 - val_loss: 1.8590 - val_acc: 0.3662
Epoch 4/25
106s - loss: 0.8502 - acc: 0.6855 - val_loss: 2.0930 - val_acc: 0.3778
Epoch 5/25
97s - loss: 0.4570 - acc: 0.8411 - val_loss: 2.5800 - val_acc: 0.3786
Epoch 6/25
99s - loss: 0.2407 - acc: 0.9239 - val_loss: 2.9577 - val_acc: 0.3704
Epoch 7/25
117s - loss: 0.1441 - acc: 0.9594 - val_loss: 3.3411 - val_acc: 0.3746
Epoch 8/25
102s - loss: 0.1048 - acc: 0.9715 - val_loss: 3.5307 - val_acc: 0.3630
Epoch 9/25
113s - loss: 0.0857 - acc: 0.9776 - val_loss: 3.5657 - val_acc: 0.3686
Epoch 10/25
128s - loss: 0.0631 - acc: 0.9840 - val_loss: 3.6607 - val_acc: 0.3776
Epoch 11/25
114s - loss: 0.0532 - acc: 0.9864 - val_loss: 3.7602 - val_acc: 0.3704
Epoch 12/25
94s - loss: 0.0467 - acc: 0.9883 - val_loss: 3.9713 - val_acc: 0.3698
Epoch 13/25
95s - loss: 0.0442 - acc: 0.9887 - val_loss: 3.9533 - val_acc: 0.3708
Epoch 14/25
112s - loss: 0.0354 - acc: 0.9915 - val_loss: 4.1183 - val_acc: 0.3672
Epoch 15/25
108s - loss: 0.0356 - acc: 0.9914 - val_loss: 4.0347 - val_acc: 0.3786
Epoch 16/25
114s - loss: 0.0387 - acc: 0.9900 - val_loss: 4.2278 - val_acc: 0.3728
Epoch 17/25
113s - loss: 0.0392 - acc: 0.9902 - val_loss: 4.1548 - val_acc: 0.3616
Epoch 18/25
94s - loss: 0.0374 - acc: 0.9901 - val_loss: 4.2827 - val_acc: 0.3704
Epoch 19/25
91s - loss: 0.0376 - acc: 0.9900 - val_loss: 4.2222 - val_acc: 0.3684
Epoch 20/25
91s - loss: 0.0303 - acc: 0.9926 - val_loss: 4.3240 - val_acc: 0.3680
Epoch 21/25
91s - loss: 0.0301 - acc: 0.9920 - val_loss: 4.2844 - val_acc: 0.3704
Epoch 22/25
91s - loss: 0.0287 - acc: 0.9925 - val_loss: 4.4545 - val_acc: 0.3760
Epoch 23/25
92s - loss: 0.0277 - acc: 0.9928 - val_loss: 4.4771 - val_acc: 0.3722
Epoch 24/25
92s - loss: 0.0275 - acc: 0.9926 - val_loss: 4.4285 - val_acc: 0.3676
Epoch 25/25
97s - loss: 0.0307 - acc: 0.9918 - val_loss: 4.3401 - val_acc: 0.3790
Three layer Neural Network Accuracy: 37.90%

```

Figure 5: The Screenshots of the result of **three hidden layer** neural network: input:3072, first hidden layer: 3072 nodes, second hidden layer: 128 nodes, third hidden layer: 64 nodes, output:10 nodes. 25 epochs is used here.

From the result we can get the conclusion below:

- The result of the three layers network is not improved and even a bit worse than the two layer network. The reason may be the parameters are not well chosen such as the number of nodes per layer. At the same time, it shows that for the choosing of the deep learning parameters, doing the experiments to choose a better model is an important technique to achieve a better performance.

4 Extra submission

This part is the extra submission for this problem by using the CNN(Convolutional Neural Networks) algorithm.

The original training data of the color image is stored in a 3072 1-D array. In order to do the CNN, it has to be reshaped to the size of $3 * 32 * 32$, where 3 is the three color(RGB) channel, 32 is the number of row pixel and the other 32 is the number of the column pixel. In order to verify whether the data is correctly reshaped, the first 9 images are plotted and shown in figure 6. From the result, we can see the image is not clearly plotted. I printed out the data and found the range of the value of each pixel is not from 0 to 255. It is also not from 0 to 1 as a normalized version of the 0 to 255. It contains some negative value and some positive value. The pixel value of the first image before reshape and after reshape is given in figure 7.

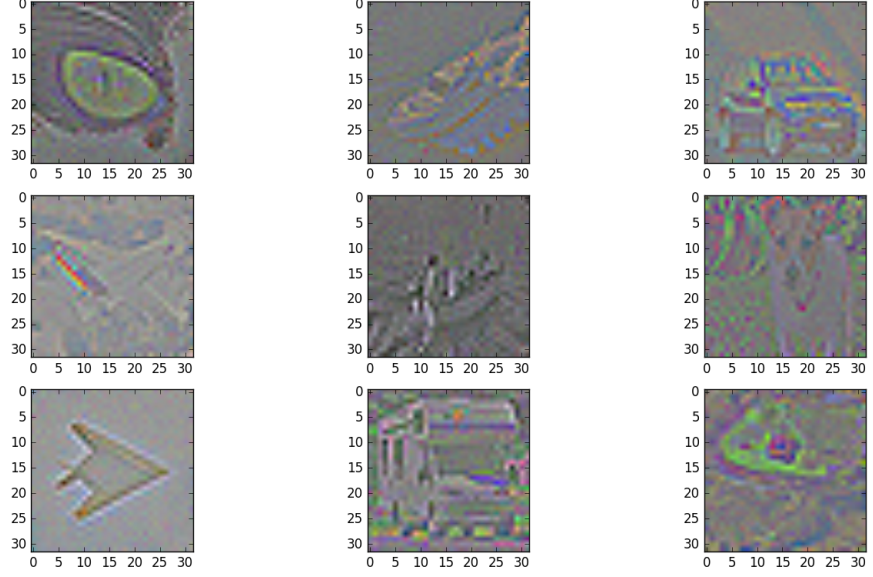


Figure 6: The plot result of the first 9 images. The original 3072 vector are reshaped to $3 * 32 * 32$. To verify whether the image is correctly reshaped, the plotted result is shown here.

```

('X_train[0] shape before reshape', (3072,))
('X_train[0] before reshape', array([-2.33455133,  0.25468308, -0.66863477, ..., -0.337827 ,
    -0.5848341 , -0.66505492], dtype=float32))
('X_train[0] shape after reshape', (3, 32, 32))
('X_train[0] after reshape', array([[[ -2.33455133e+00,  2.54683077e-01, -6.68634772e-01, ...,
    4.29762459e+00, -1.87078702e+00,  3.90259838e+00],
    [ 4.95189935e-01, -5.64029634e-01,  2.54760116e-01, ...,
    3.04553121e-01, -2.47317757e-02,  4.10869747e-01],
    [ 4.63162005e-01, -4.39032465e-01,  1.86107054e-01, ...,
    3.16653919e+00,  9.44282234e-01,  2.41789508e+00],
    ...,
    [ 3.48544151e-01, -5.81714511e-01,  3.75492036e-01, ...,
    6.16347373e-01,  9.03050542e-01,  2.21743211e-01],
    [ 6.78724289e-01,  6.45941615e-01, -2.95581579e-01, ...,
    3.26629402e-03, -3.63977909e-01,  1.35533243e-01],
    [ 1.24035609e+00,  1.80171877e-01,  4.89222527e-01, ...,
    1.19894035e-01,  1.26535660e-02,  2.78778553e-01]],
    [[ -8.79337847e-01, -8.33894014e-01,  7.36563146e-01, ...,
    5.07737112e+00, -2.23829460e+00,  2.73643589e+00],
    [ -6.56889856e-01,  2.46842965e-01, -1.69128633e+00, ...,
    -1.16178131e+00,  9.91790649e-03,  2.46455264e+00],
    [ 7.92051330e-02, -2.65118897e-01, -3.05299193e-01, ...,
    2.13135552e+00,  5.81940189e-02,  1.85456562e+00],
    ...,
    [ -2.18744740e-01, -3.61516438e-02,  8.35463583e-01, ...,
    8.65817249e-01,  1.16681017e-01, -2.45418772e-01],
    [ 1.13289237e+00,  5.19839048e-01,  4.72078651e-01, ...,
    7.65660226e-01,  7.40637407e-02, -1.19368303e+00],
    [ 1.04927766e+00, -4.54853997e-02,  3.90172243e-01, ...,
    3.51632029e-01, -6.50370836e-01, -7.95238018e-01]],
    [[ -5.61994314e-01, -7.08573237e-02, -1.52898639e-01, ...,
    4.81897640e+00, -3.19537044e+00,  3.51406002e+00],
    [ -3.23754817e-01,  3.45964253e-01,  1.98289499e-01, ...,
    -1.01577687e+00,  2.69423842e-01,  9.98487175e-01],
    [ -7.17723489e-01, -1.92791391e-02, -3.74344885e-01, ...,
    3.44630265e+00,  7.53066763e-02,  1.52950501e+00],

```

Figure 7: The pixel value of the first image before reshape and after reshape.

As from the plotted result, the image can still be recognized such as the top right is a vehicle and the left bottom is an airplane. The data is used for the CNN algorithm to check the performance.

4.1 Small size CNN

The configuration of this small size CNN is given in figure 8.

The accuracy of this CNN network is **68.98%** under 25 epochs(Screenshots of the result is given in figure 9).

The code of this part is provided in the file

`big_data_assignment3_cnn.py`.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 32)	896
dropout_1 (Dropout)	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_1 (MaxPooling2D)	(None, 32, 16, 16)	0
flatten_1 (Flatten)	(None, 8192)	0
dense_1 (Dense)	(None, 512)	4194816
dropout_2 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 10)	5130
Total params: 4,210,090		
Trainable params: 4,210,090		
Non-trainable params: 0		

Figure 8: The summary of the large CNN network

```

Train on 45000 samples, validate on 5000 samples
Epoch 1/25
45000/45000 [=====] - 615s - loss: 1.5809 - acc: 0.4411 - val_loss: 1.2245 - val_acc: 0.5790
Epoch 2/25
45000/45000 [=====] - 580s - loss: 1.1520 - acc: 0.6011 - val_loss: 1.0994 - val_acc: 0.6252
Epoch 3/25
45000/45000 [=====] - 591s - loss: 0.9478 - acc: 0.6706 - val_loss: 1.0181 - val_acc: 0.6514
Epoch 4/25
45000/45000 [=====] - 576s - loss: 0.8006 - acc: 0.7208 - val_loss: 0.9868 - val_acc: 0.6688
Epoch 5/25
45000/45000 [=====] - 721s - loss: 0.6814 - acc: 0.7625 - val_loss: 0.9992 - val_acc: 0.6794
Epoch 6/25
45000/45000 [=====] - 697s - loss: 0.5818 - acc: 0.7943 - val_loss: 1.0378 - val_acc: 0.6796
Epoch 7/25
45000/45000 [=====] - 705s - loss: 0.4861 - acc: 0.8311 - val_loss: 1.0896 - val_acc: 0.6790
Epoch 8/25
45000/45000 [=====] - 697s - loss: 0.4044 - acc: 0.8566 - val_loss: 1.1417 - val_acc: 0.6722
Epoch 9/25
45000/45000 [=====] - 681s - loss: 0.3509 - acc: 0.8769 - val_loss: 1.1743 - val_acc: 0.6838
Epoch 10/25
45000/45000 [=====] - 601s - loss: 0.3050 - acc: 0.8924 - val_loss: 1.1905 - val_acc: 0.6836
Epoch 11/25
45000/45000 [=====] - 581s - loss: 0.2635 - acc: 0.9081 - val_loss: 1.2124 - val_acc: 0.6848
Epoch 12/25
45000/45000 [=====] - 575s - loss: 0.2364 - acc: 0.9188 - val_loss: 1.2665 - val_acc: 0.6862
Epoch 13/25
45000/45000 [=====] - 573s - loss: 0.2084 - acc: 0.9284 - val_loss: 1.3295 - val_acc: 0.6884
Epoch 14/25
45000/45000 [=====] - 573s - loss: 0.1864 - acc: 0.9364 - val_loss: 1.3739 - val_acc: 0.6842
Epoch 15/25
45000/45000 [=====] - 573s - loss: 0.1689 - acc: 0.9411 - val_loss: 1.4176 - val_acc: 0.6850
Epoch 16/25
45000/45000 [=====] - 574s - loss: 0.1559 - acc: 0.9468 - val_loss: 1.4050 - val_acc: 0.6830
Epoch 17/25
45000/45000 [=====] - 590s - loss: 0.1453 - acc: 0.9502 - val_loss: 1.4480 - val_acc: 0.6822
Epoch 18/25
45000/45000 [=====] - 610s - loss: 0.1346 - acc: 0.9538 - val_loss: 1.4806 - val_acc: 0.6858
Epoch 19/25
45000/45000 [=====] - 585s - loss: 0.1246 - acc: 0.9582 - val_loss: 1.5170 - val_acc: 0.6864
Epoch 20/25
45000/45000 [=====] - 577s - loss: 0.1150 - acc: 0.9610 - val_loss: 1.5386 - val_acc: 0.6844
Epoch 21/25
45000/45000 [=====] - 639s - loss: 0.1110 - acc: 0.9625 - val_loss: 1.5556 - val_acc: 0.6842
Epoch 22/25
45000/45000 [=====] - 622s - loss: 0.1036 - acc: 0.9655 - val_loss: 1.5666 - val_acc: 0.6842
Epoch 23/25
45000/45000 [=====] - 618s - loss: 0.0973 - acc: 0.9676 - val_loss: 1.5782 - val_acc: 0.6884
Epoch 24/25
45000/45000 [=====] - 656s - loss: 0.0915 - acc: 0.9698 - val_loss: 1.5746 - val_acc: 0.6862
Epoch 25/25
45000/45000 [=====] - 682s - loss: 0.0880 - acc: 0.9715 - val_loss: 1.6218 - val_acc: 0.6898
CNN Accuracy: 68.98%

```

Figure 9: The result of the large CNN network for 25 epochs

4.2 Large size CNN

The configuration of this large size CNN is given in figure 10.

The accuracy of this CNN network is **82.40%** under 25 epochs(Screenshots of the result is given in figure 11).

The code of this part is provided in the file

`big_data_assignment3_cnn_large.py`.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 32)	896
dropout_1 (Dropout)	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_1 (MaxPooling2D)	(None, 32, 16, 16)	0
conv2d_3 (Conv2D)	(None, 64, 16, 16)	18496
dropout_2 (Dropout)	(None, 64, 16, 16)	0
conv2d_4 (Conv2D)	(None, 64, 16, 16)	36928
max_pooling2d_2 (MaxPooling2D)	(None, 64, 8, 8)	0
conv2d_5 (Conv2D)	(None, 128, 8, 8)	73856
dropout_3 (Dropout)	(None, 128, 8, 8)	0
conv2d_6 (Conv2D)	(None, 128, 8, 8)	147584
max_pooling2d_3 (MaxPooling2D)	(None, 128, 4, 4)	0
flatten_1 (Flatten)	(None, 2048)	0
dropout_4 (Dropout)	(None, 2048)	0
dense_1 (Dense)	(None, 1024)	2098176
dropout_5 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 512)	524800
dropout_6 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 10)	5130
Total params: 2,915,114		
Trainable params: 2,915,114		
Non-trainable params: 0		

Figure 10: The summary of the large CNN network

```

(y_train.shape, (45000, 10))
Train on 45000 samples, validate on 5000 samples
Epoch 1/25
45000/45000 [=====] - 987s - loss: 1.8779 - acc: 0.2988 - val_loss: 1.9462 - val_acc: 0.3152
Epoch 2/25
45000/45000 [=====] - 974s - loss: 1.3629 - acc: 0.5085 - val_loss: 1.1640 - val_acc: 0.5844
Epoch 3/25
45000/45000 [=====] - 972s - loss: 1.0803 - acc: 0.6116 - val_loss: 1.0649 - val_acc: 0.6248
Epoch 4/25
45000/45000 [=====] - 972s - loss: 0.9251 - acc: 0.6730 - val_loss: 0.8527 - val_acc: 0.6958
Epoch 5/25
45000/45000 [=====] - 971s - loss: 0.8199 - acc: 0.7092 - val_loss: 0.8662 - val_acc: 0.6970
Epoch 6/25
45000/45000 [=====] - 971s - loss: 0.7464 - acc: 0.7379 - val_loss: 0.7365 - val_acc: 0.7418
Epoch 7/25
45000/45000 [=====] - 971s - loss: 0.6861 - acc: 0.7566 - val_loss: 0.7218 - val_acc: 0.7454
Epoch 8/25
45000/45000 [=====] - 971s - loss: 0.6414 - acc: 0.7726 - val_loss: 0.7018 - val_acc: 0.7556
Epoch 9/25
45000/45000 [=====] - 973s - loss: 0.5886 - acc: 0.7917 - val_loss: 0.6570 - val_acc: 0.7748
Epoch 10/25
45000/45000 [=====] - 987s - loss: 0.5570 - acc: 0.8030 - val_loss: 0.6343 - val_acc: 0.7770
Epoch 11/25
45000/45000 [=====] - 992s - loss: 0.5275 - acc: 0.8124 - val_loss: 0.5979 - val_acc: 0.7930
Epoch 12/25
45000/45000 [=====] - 1000s - loss: 0.4989 - acc: 0.8243 - val_loss: 0.6009 - val_acc: 0.7898
Epoch 13/25
45000/45000 [=====] - 1139s - loss: 0.4741 - acc: 0.8309 - val_loss: 0.5912 - val_acc: 0.7950
Epoch 14/25
45000/45000 [=====] - 1088s - loss: 0.4538 - acc: 0.8387 - val_loss: 0.5712 - val_acc: 0.8008
Epoch 15/25
45000/45000 [=====] - 982s - loss: 0.4310 - acc: 0.8469 - val_loss: 0.5693 - val_acc: 0.8028
Epoch 16/25
45000/45000 [=====] - 974s - loss: 0.4174 - acc: 0.8508 - val_loss: 0.5906 - val_acc: 0.8016
Epoch 17/25
45000/45000 [=====] - 973s - loss: 0.3920 - acc: 0.8591 - val_loss: 0.5444 - val_acc: 0.8122
Epoch 18/25
45000/45000 [=====] - 978s - loss: 0.3774 - acc: 0.8657 - val_loss: 0.5646 - val_acc: 0.8052
Epoch 19/25
45000/45000 [=====] - 990s - loss: 0.3661 - acc: 0.8695 - val_loss: 0.5509 - val_acc: 0.8164
Epoch 20/25
45000/45000 [=====] - 975s - loss: 0.3523 - acc: 0.8761 - val_loss: 0.5416 - val_acc: 0.8150
Epoch 21/25
45000/45000 [=====] - 974s - loss: 0.3345 - acc: 0.8803 - val_loss: 0.5356 - val_acc: 0.8216
Epoch 22/25
45000/45000 [=====] - 973s - loss: 0.3247 - acc: 0.8834 - val_loss: 0.5357 - val_acc: 0.8184
Epoch 23/25
45000/45000 [=====] - 972s - loss: 0.3149 - acc: 0.8864 - val_loss: 0.5467 - val_acc: 0.8166
Epoch 24/25
45000/45000 [=====] - 973s - loss: 0.3066 - acc: 0.8892 - val_loss: 0.5531 - val_acc: 0.8154
Epoch 25/25
45000/45000 [=====] - 974s - loss: 0.2920 - acc: 0.8941 - val_loss: 0.5313 - val_acc: 0.8240
CNN large Accuracy: 82.40%

```

Figure 11: The result of the large CNN network for 25 epochs

4.3 Conclusion

- The CNN can have a better performance than the simply single or multiple layer neural network as it can collect features from the data set.
- The large size CNN can have a better performance than the small size CNN.