Final results: We classified handwritten numeric data (MNIST) by SVM

Procedure

① First, we've downloaded MNIST data

② Then, we wrote binary file of MNIST to CSV

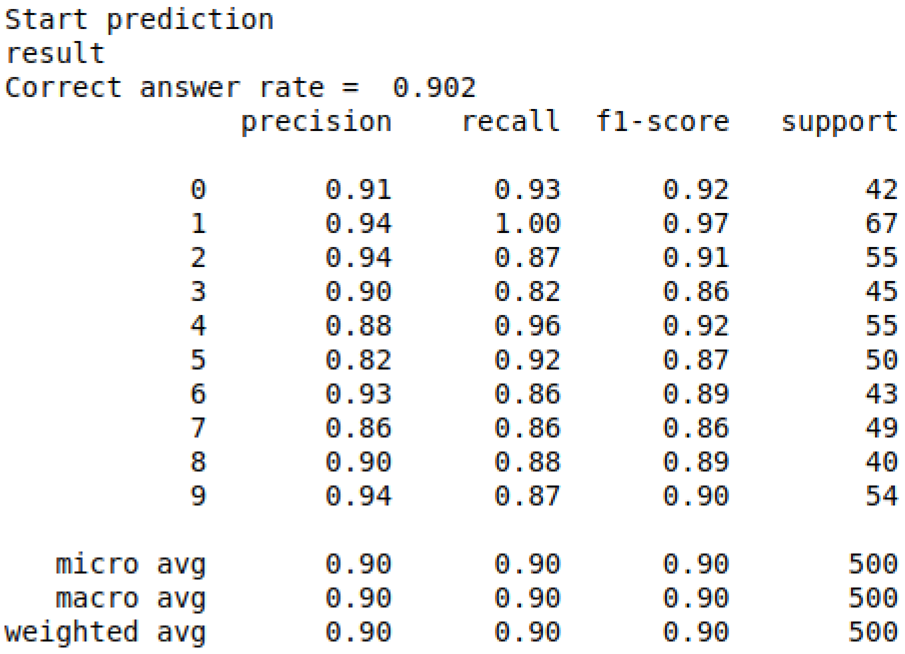
(I couldn't find the library which it does easily, and since we weren't using Neural networks, I've decided not to use tensorflow&keros as we did while playing with MNIST in class

③ We've written down the CSV data to the image data and to check whether the CSV can properly write it out

④ We've did ML and predictions with SVM

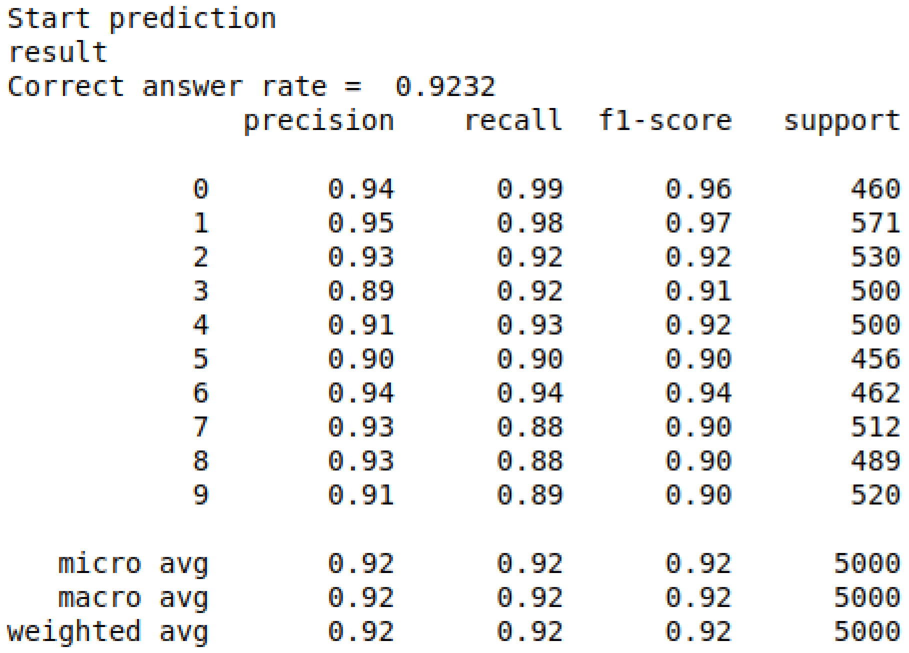
Results:

1) For the first trial, when we were using 5k training data and 500 test data, we've got the accuracy of **0.9**



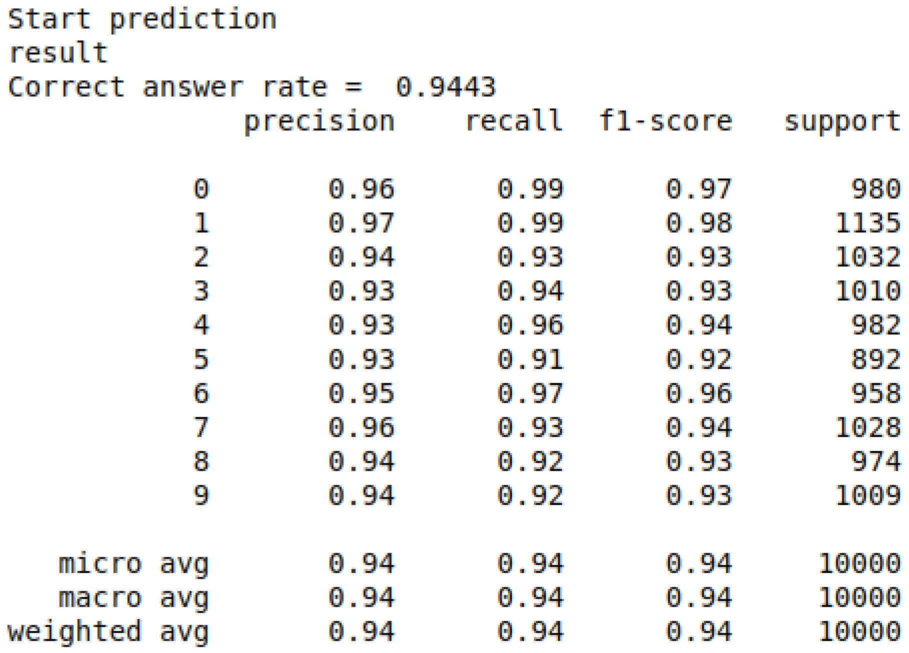
```
Start prediction
result
Correct answer rate =  0.902
              precision    recall  f1-score   support

           0       0.91      0.93      0.92        42
           1       0.94      1.00      0.97        67
           2       0.94      0.87      0.91        55
           3       0.90      0.82      0.86        45
           4       0.88      0.96      0.92        55
           5       0.82      0.92      0.87        50
           6       0.93      0.86      0.89        43
           7       0.86      0.86      0.86        49
           8       0.90      0.88      0.89        40
           9       0.94      0.87      0.90        54

   micro avg       0.90      0.90      0.90       500
   macro avg       0.90      0.90      0.90       500
weighted avg       0.90      0.90      0.90       500
```

2)Then we've decided to increase the data sets to 50k training data and 5k test data. The accuracy rose to **0.92**



```
Start prediction
result
Correct answer rate =  0.9232
                  precision    recall  f1-score   support

              0       0.94      0.99      0.96       460
              1       0.95      0.98      0.97       571
              2       0.93      0.92      0.92       530
              3       0.89      0.92      0.91       500
              4       0.91      0.93      0.92       500
              5       0.90      0.90      0.90       456
              6       0.94      0.94      0.94       462
              7       0.93      0.88      0.90       512
              8       0.93      0.88      0.90       489
              9       0.91      0.89      0.90       520

   micro avg       0.92      0.92      0.92      5000
   macro avg       0.92      0.92      0.92      5000
weighted avg       0.92      0.92      0.92      5000
```

3) Lastly we've check for the data with 60k and test 10k, and the accuracy rose till **94.5**

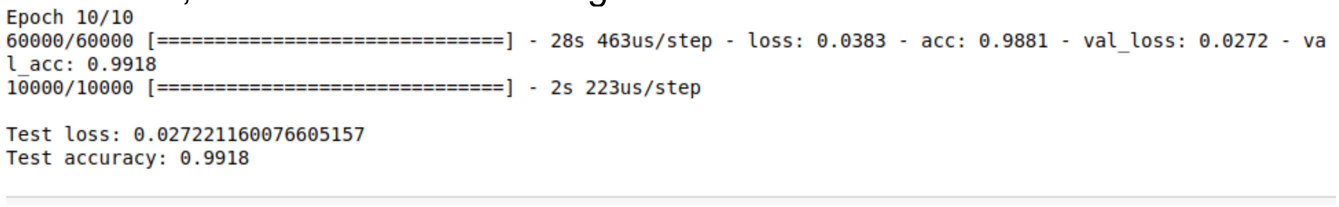


```
Start prediction
result
Correct answer rate =  0.9443
                  precision    recall  f1-score   support

              0       0.96      0.99      0.97       980
              1       0.97      0.99      0.98      1135
              2       0.94      0.93      0.93      1032
              3       0.93      0.94      0.93      1010
              4       0.93      0.96      0.94       982
              5       0.93      0.91      0.92       892
              6       0.95      0.97      0.96       958
              7       0.96      0.93      0.94      1028
              8       0.94      0.92      0.93       974
              9       0.94      0.92      0.93      1009

   micro avg       0.94      0.94      0.94     10000
   macro avg       0.94      0.94      0.94     10000
weighted avg       0.94      0.94      0.94     10000
```

Comparing to the other model:

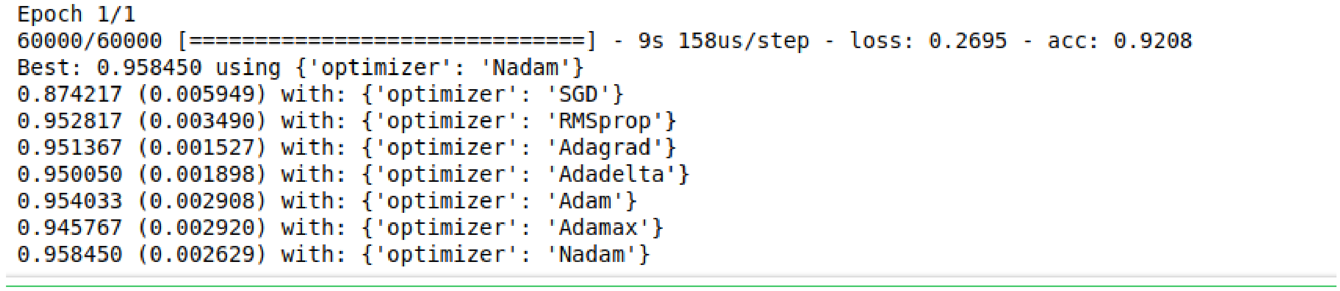
1) When during the semester we've used the Convolutional Neural Net

with keras, we’ve seen results as high as **0.98-0.99**



```
Epoch 10/10
60000/60000 [==============================] - 28s 463us/step - loss: 0.0383 - acc: 0.9881 - val_loss: 0.0272 - va
l_acc: 0.9918
10000/10000 [==============================] - 2s 223us/step

Test loss: 0.027221160076605157
Test accuracy: 0.9918
```

2) In general, trying different optimizers, we’ve seen the next results



```
Epoch 1/1
60000/60000 [==============================] - 9s 158us/step - loss: 0.2695 - acc: 0.9208
Best: 0.958450 using {'optimizer': 'Nadam'}
0.874217 (0.005949) with: {'optimizer': 'SGD'}
0.952817 (0.003490) with: {'optimizer': 'RMSprop'}
0.951367 (0.001527) with: {'optimizer': 'Adagrad'}
0.950050 (0.001898) with: {'optimizer': 'Adadelta'}
0.954033 (0.002908) with: {'optimizer': 'Adam'}
0.945767 (0.002920) with: {'optimizer': 'Adamax'}
0.958450 (0.002629) with: {'optimizer': 'Nadam'}
```

Conclusion:

So, even we obviously see that DNN, CNN can work better with MNIST data, not so sophisticated SVM approach still can give you quite good results, especially if we can increase the number of train data points to the whole MNIST database.