COGS 260: Assignment 2

Aditya Verma
A53219148
Department of Electrical and Computer Engineering
University of California, San Diego
`arv018@ucsd.edu`

Abstract

In this assignment, we experiment with different classification techniques for MNIST digits like K Nearest neighbor, SVM, and Neural Networks. The focus of this assignment is to experiment with all different techniques, libraries the reference to all is given at the end.

1 1-Nearest Neighbor

1.1 Dataset

1-Nearest Neighbor is computationally very expensive and slow process, so I reduced the training and test data size.

- Train Size: 20000
- Test Size: 10000

Table 1: 1-Nearest neighbor classifier settings

Number of neighbors	1
Algorithm	KdTree
Weight	Uniform
Distance Metric	Euclidean
Leaf size	30

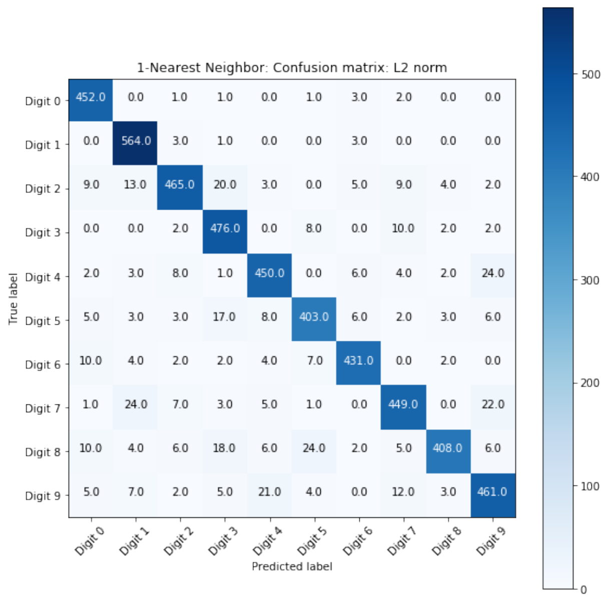


Figure 1: Test data confusion matrix for 1-Nearest Neighbor classifier

1.2 Results & Discussion

With the above mentioned setting we achieve the below mentioned results:

- Test accuracy: 91.18%
- Time taken: 400s

As can be observed, the accuracy is descent enough but less than what we can achieve with other techniques as we shall study ahead in this report. The biggest challenge with 1-Nearest neighbor classifier is the slow speed. With increasing the size of training size, the test time exceeds exponentially. For this reason, we had to reduce the training and test data size.

2 Support Vector Machine

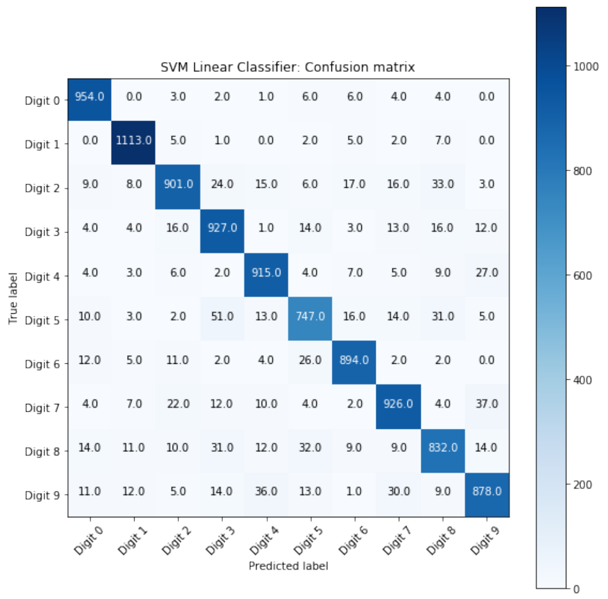
2.1 Linear Classifier

For SVM, we use a larger dataset over 1-Nearest neighbor.

- Train size: 50000
- Test size: 10000

Table 2: SVM classifier settings

Loss function	Hinge loss
Penalty	L2 norm
Number of epocs	5
Cache size	200



	Digit 0	Digit 1	Digit 2	Digit 3	Digit 4	Digit 5	Digit 6	Digit 7	Digit 8	Digit 9
Digit 0	954.0	0.0	3.0	2.0	1.0	6.0	6.0	4.0	4.0	0.0
Digit 1	0.0	1113.0	5.0	1.0	0.0	2.0	5.0	2.0	7.0	0.0
Digit 2	9.0	8.0	901.0	24.0	15.0	6.0	17.0	16.0	33.0	3.0
Digit 3	4.0	4.0	16.0	927.0	1.0	14.0	3.0	13.0	16.0	12.0
Digit 4	4.0	3.0	6.0	2.0	915.0	4.0	7.0	5.0	9.0	27.0
Digit 5	10.0	3.0	2.0	51.0	13.0	747.0	16.0	14.0	31.0	5.0
Digit 6	12.0	5.0	11.0	2.0	4.0	26.0	894.0	2.0	2.0	0.0
Digit 7	4.0	7.0	22.0	12.0	10.0	4.0	2.0	926.0	4.0	37.0
Digit 8	14.0	11.0	10.0	31.0	12.0	32.0	9.0	9.0	832.0	14.0
Digit 9	11.0	12.0	5.0	14.0	36.0	13.0	1.0	30.0	9.0	878.0

Figure 2: Linear SVM: Confusion matrix

2.2 Result & Discussion

- Train accuracy: 92.23%
- Test accuracy: 90.87%
- Time to build SVM classifier: 425s

Linear SVM gives better accuracy over k-nearest neighbor. The only time taken in this technique is in trainning the SVM which is observe to be 450s. The test time is almost instantaneous and good for real time practical usage.

3 Spatial Pyramid Matching

3.1 Description

This question is broken down into two parts. First we extract the SIFT features for the training dataset. From these features, we build a SVM classifier. SIFT features are extracted using OpenCV inbuilt functionn:

sift = cv2.xfeatures2d.SIFT˙create()

We use the scikit-learn in-build SVM classifier, and the optimized parameter are determined using grid search.

- Training epocs: 10
- Pyramids used: 3
- Patch size: 16
- Dictionary size: 200

3.2 Result & Discussion

- Train Accuracy: 87.23%
- Test Accuracy: 86.87%

Table 3: Spatial pyramid matching: report-summary

SrNo.	precision	recall	f1-score	Support
0	0.90	0.93	0.92	980
1	0.97	0.98	0.98	1135
2	0.86	0.85	0.86	1032
3	0.75	0.79	0.77	1010
4	0.92	0.88	0.90	982
5	0.80	0.75	0.77	892
6	0.94	0.94	0.94	958
7	0.83	0.86	0.84	1028
8	0.75	0.71	0.73	974
9	0.77	0.79	0.78	1009
avg / total	0.85	0.85	0.85	10000

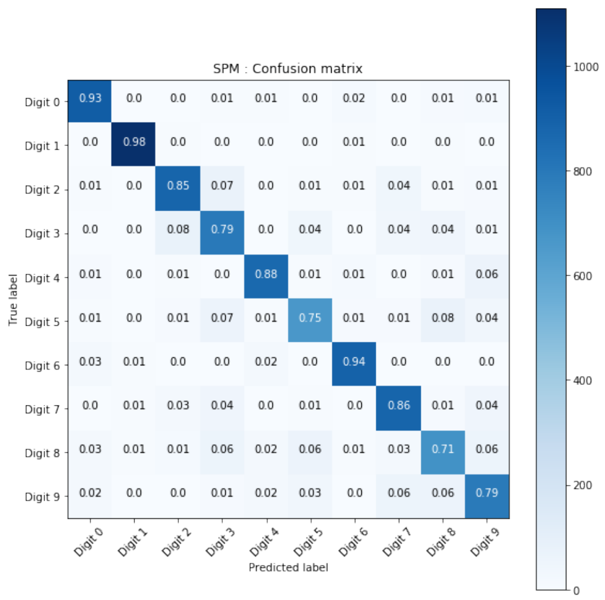


Figure 3: Spatial pyramid matching: Confusion matrix

With the SIFT feature extractor and SVM based classification, we did not get the best possible results. This could be due to the hyperparameters not being fine tuned to the best possible configuration.

4 Convolutional Neural Network

4.1 Description

The task undertaken in this part of the question is to classify the digits using Convolutional neural network. In all the techniques implemented above, we treated the digits as a 784 dimensional row vector which did not exploit the spatial information contained in images. In this part, we convert the row vector into 28x28 matrix, and exploit spatial information to classify digits. The CNN filters extract informations from each layer, which lead to high level abstraction of digits and later help classify them.

Table 4: Convolutional neural network architecture

Layer	Type	Size	Stride	Padding	Activation
Input Layer	28x28x1 Input Image				
Layer 2	Convolution layer	5x5x1x32.	(1,1)	Same	relu
Layer 3	Maxpool layer	2x2.	(2,2)		
Layer 4	Convolution layer	5x5x32x64.	(1,1)	Same	relu
Layer 5	Maxpool layer	2x2	(1,1)		
Layer 6	Fully Connected	(7x7x64,1024).			relu
Layer 7	FC. output layer	(1024,10).			softmax

Table 5: CNN training configuration

Hyperparameter	Value
Optimizer	Rmsprop
Number of epocs	500
Minibatch size	128
Dropout ratio	0.3

4.2 Result and Discussion

- Train Accuracy: 96.2%
- Test Accuracy: 95.2%

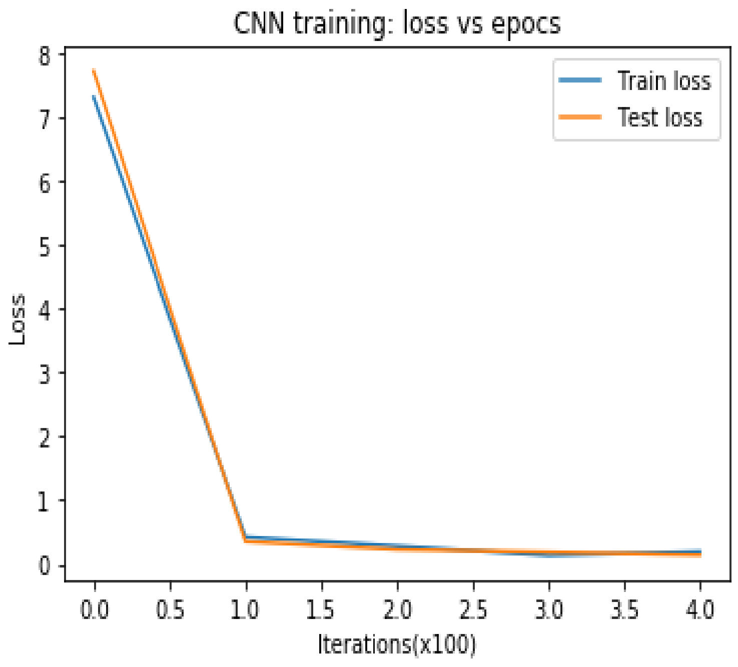
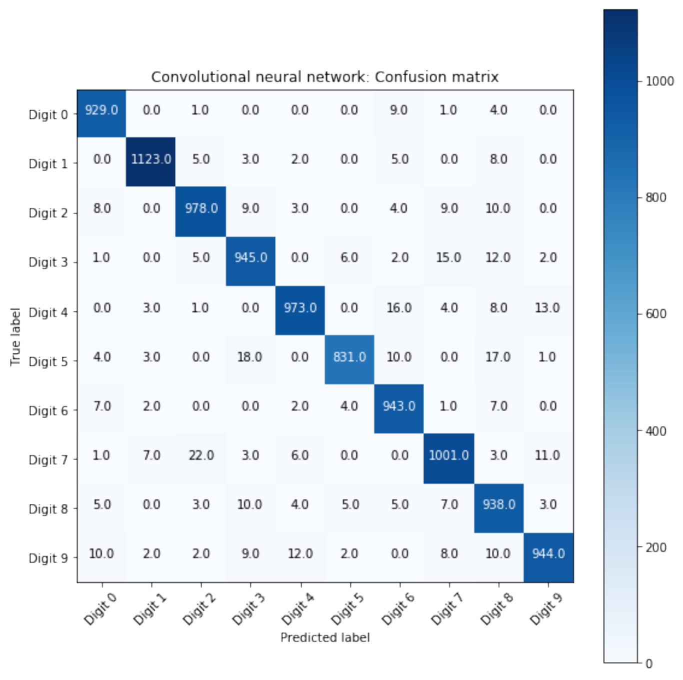


Figure 4: CNN training: Plot of training loss vs epocs



	Digit 0	Digit 1	Digit 2	Digit 3	Digit 4	Digit 5	Digit 6	Digit 7	Digit 8	Digit 9
Digit 0	929.0	0.0	1.0	0.0	0.0	0.0	9.0	1.0	4.0	0.0
Digit 1	0.0	1123.0	5.0	3.0	2.0	0.0	5.0	0.0	8.0	0.0
Digit 2	8.0	0.0	978.0	9.0	3.0	0.0	4.0	9.0	10.0	0.0
Digit 3	1.0	0.0	5.0	945.0	0.0	6.0	2.0	15.0	12.0	2.0
Digit 4	0.0	3.0	1.0	0.0	973.0	0.0	16.0	4.0	8.0	13.0
Digit 5	4.0	3.0	0.0	18.0	0.0	831.0	10.0	0.0	17.0	1.0
Digit 6	7.0	2.0	0.0	0.0	2.0	4.0	943.0	1.0	7.0	0.0
Digit 7	1.0	7.0	22.0	3.0	6.0	0.0	0.0	1001.0	3.0	11.0
Digit 8	5.0	0.0	3.0	10.0	4.0	5.0	5.0	7.0	938.0	3.0
Digit 9	10.0	2.0	2.0	9.0	12.0	2.0	0.0	8.0	10.0	944.0

Figure 5: CNN test: confusion matrix

5 Deep Belief Nets

In this question, we take a very different approach over all the approaches taken above. All the techniques utilized above were discriminative in nature to classify the input data into different classes. In this part, we exploit Restricted Boltzmann machine and deep belief nets.

- First hidden input layer is the input layer of 784 dimension
- Fully ully connected dense hidden layers have 500,200,50 neurons respectively. Activation function: 'relu'
- Output visible layer has 10 neurons.

5.1 Training

Training the network can be broken down into two phases:-

- The network is trained in the form of restricted boltzmann machine with unsupervised network. Any two pair of layers are trained. The pairs are taken progressively, from the input neurons towards the output neurons.
- After all the three layers are trained, we train the overall network with backpropagation.
- It is expected that this two step process will give a better trained network compared to training the overall network in a single go and converge at global minima.

5.2 Result & Discussion

Overall trained network accuracies

- Train Accuracy: 99.72%
- Test Accuracy: 98.23%

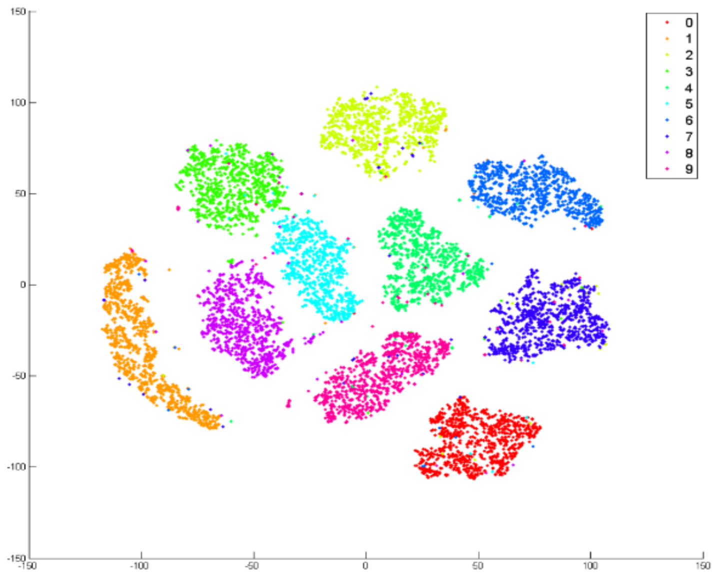


Figure 6: TSNE visualization of last hidden layer activations

As can be observed from the TSNE plot above, all the 10 classes as projected from the last hidden layer as well separable.This shows that the Deep belief network we created above has a very good classification accuracy.

6 Source code

All the codes for this assignment are avaialbe at MNIST˙Solution

To accomplish this assignment, I have used several Scikit in-built functionalities, Tensorflow libraries, and open source repositories.

The reference to every open source library is given in the github repository at the link given above