COGS 260: Assignment 2

Saurabh Gupta
A53091070
sag043@ucsd.edu

Abstract

In this assignment, we compare the performance of different image classification algorithms on MNIST dataset. Emphasis is given on using different tools and techniques which can run different algorithms efficiently. As a result, golang, torch, python and MATLAB were used for this assignment depending on the problem.

1 1-Nearest Neighbor

1.1 Experiments

1.1.1 Choice of distance function

L1 vs L2 - It is interesting to consider differences between the two metrics. In particular, the L2 distance is much more unforgiving than the L1 distance when it comes to differences between two vectors. That is, the L2 distance prefers many medium disagreements to one big one.

1.1.2 Bailout

This implementation of Nearest Neighbours in golang involved computation of L2 distances using "for loops". Each loop calculated distance for one dimension and these distances were subsequently added to the accumulator to calculate the final L2 distance. Bailout refers to the technique of breaking the for loop when we know that the current accumulation of L2 distances has already crossed the so far shortest distance. This technique helped in speeding up the runtime significantly.

1.1.3 Parallelization using Goroutines

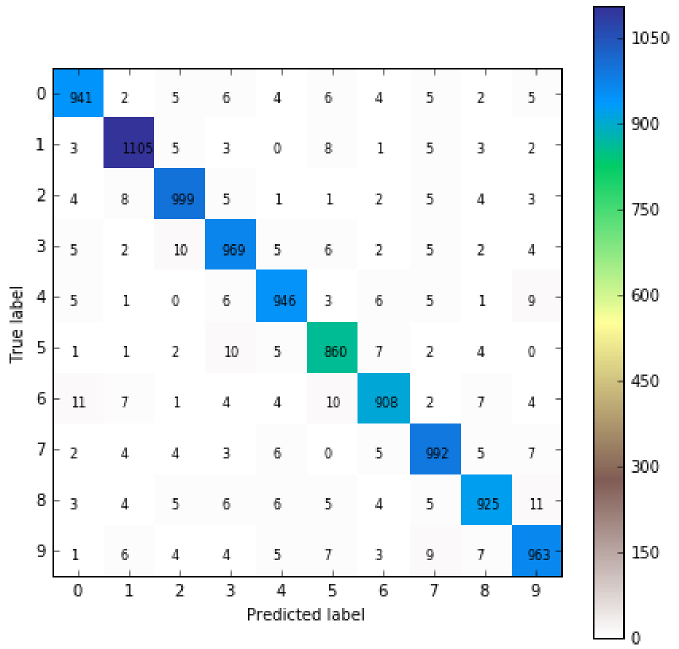
Go provides an easy way of parallel computations using light-weight goroutines. All the instances in test set were run in parallel for classification.

1.2 Results and Discussions

Algorithm	Accuracy on test set	Running Time
L2 with bailout	97.45%	403.74 s
L2 without bailout	97.45%	883.81 s
L1 without bailout	96.86%	890.1 s

Table 1: Performance results for 1-NN

Hence, the L2 distance function with bailout gave the best performance both in terms of speed and accuracy. The confusion matrix for this algorithm is shown below:



	0	1	2	3	4	5	6	7	8	9
0	941	2	5	6	4	6	4	5	2	5
1	3	1105	5	3	0	8	1	5	3	2
2	4	8	999	5	1	1	2	5	4	3
3	5	2	10	969	5	6	2	5	2	4
4	5	1	0	6	946	3	6	5	1	9
5	1	1	2	10	5	860	7	2	4	0
6	11	7	1	4	4	10	908	2	7	4
7	2	4	4	3	6	0	5	992	5	7
8	3	4	5	6	6	5	4	5	925	11
9	1	6	4	4	5	7	3	9	7	963

Figure 1: Test set confusion matrix for L2 1-Nearest Neighbor

2 Support Vector Machines

2.1 Image Data Preprocessing

All the images were centered by subtracting the mean from every feature. Further, all the images were scaled down by 255 so that every feature value is in the range [-1, 1].

2.2 Experiments and methods

Instead of using Scikit learn to perform SVM on MNIST dataset, SVM was implemented in Torch as "one fully connected layer" neural network with Multiclass Hinge loss.

Torch model definition:
model = nn.Sequential()
*model:add(nn.Reshape(28*28, true))*
*model:add(nn.Linear(28*28, 10))*
criterion = nn.MultiMarginCriterion()
model:cuda() – To run the model on GPU

Torch was specifically used so that the model could be run on GPU which provides a 10-20x speedup. The above model was optimized using Stochastic Gradient Descent (backprop) with a learning rate of 0.05 and batch size of 100.

2.3 Results and Discussions

The model just took 10 seconds to converge to a test accuracy of 91.5%. It was run for 10 epochs.

Train accuracy: 92.19%
Test accuracy: 91.58%

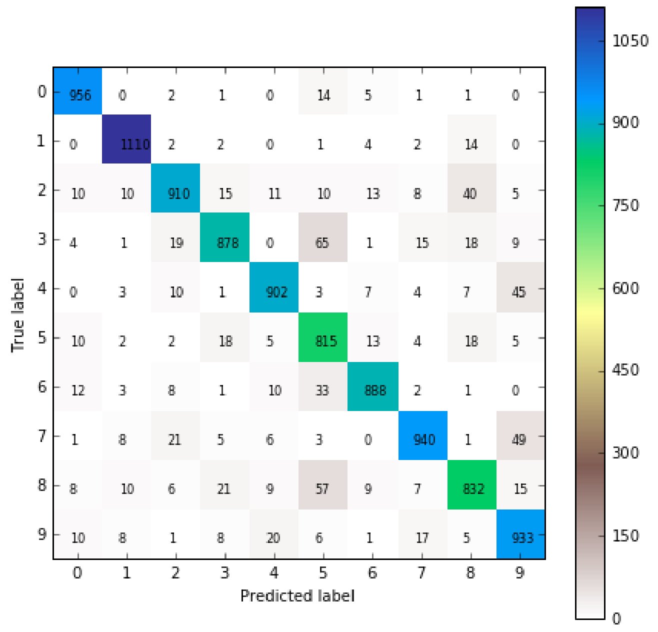


Figure 2: Test set confusion matrix for linear SVM

3 Spatial Pyramid Matching

3.1 Experiments and methods

For this problem, the MATLAB code provided by Svetlana Lazebnik was used to generate the features. Default parameters provided in the code were used.

The code first extracts SIFT descriptors on a regular grid from each image. It then runs k-means to find the dictionary. Each sift descriptor is given a label corresponding to the closest dictionary codeword. Finally, the spatial pyramid is generated from these labels.

Once these features are available, SVM (using Torch) was run as described in section 2.

Parameters used:

dictionarySize: 200

numTextonImages: 50

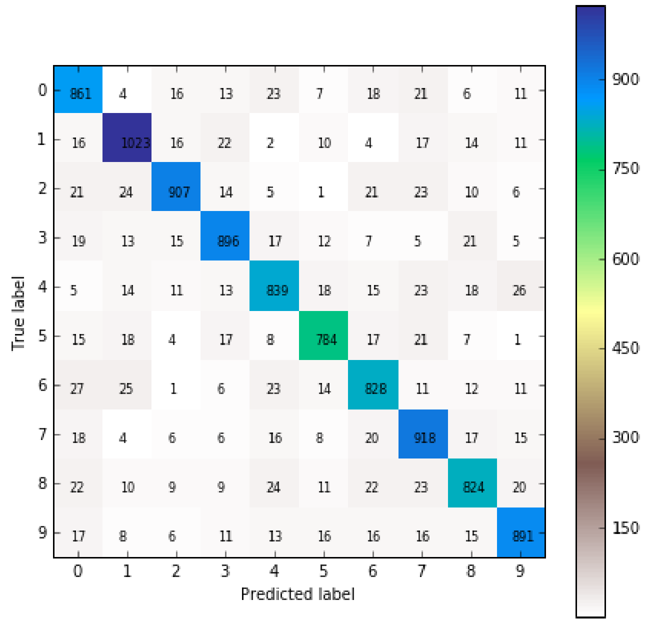
pyramidLevels: 3

patchSize: 16

3.3 Results and Discussions

Training was run for 10 epochs.

Train accuracy: 90.03%
Test accuracy: 88.51%



	0	1	2	3	4	5	6	7	8	9
0	861	4	16	13	23	7	18	21	6	11
1	16	1023	16	22	2	10	4	17	14	11
2	21	24	907	14	5	1	21	23	10	6
3	19	13	15	896	17	12	7	5	21	5
4	5	14	11	13	839	18	15	23	18	26
5	15	18	4	17	8	784	17	21	7	1
6	27	25	1	6	23	14	828	11	12	11
7	18	4	6	6	16	8	20	918	17	15
8	22	10	9	9	24	11	22	23	824	20
9	17	8	6	11	13	16	16	16	15	891

Figure 3: Test set confusion matrix for linear SVM with spatial pyramid features

4 Convolutional Neural Network

4.1 Image Data Preprocessing

All the images were centered by subtracting the mean from every feature. Further, all the images were scaled down by 255 so that every feature value is in the range [-1, 1]. As Yan LeCun's LeNet-5 was used, the images were padded by 2px on all sides to get 32px X 32px images.

4.2 Experiments and methods

LeNet-5 was implemented in Torch using CUDNN module (on GPU) to classify MNIST digits. The architecture of LeNet-5 is shown below:

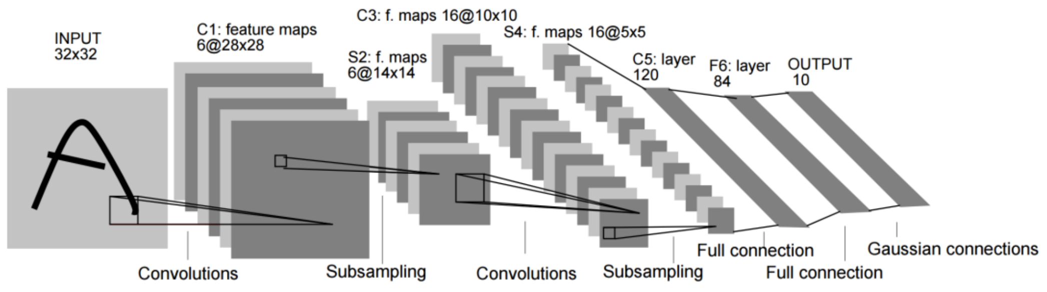


Figure 4: Architecture of LeNet-5, a CNN, used here for digit recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical

Torch model definition:

```
model = nn.Sequential()
model:add(nn.SpatialConvolutionMM(1, 32, 5, 5))
```

```
model:add(nn.Tanh())
model:add(nn.SpatialMaxPooling(3, 3, 3, 3))
-- stage 2 : mean suppresion -> filter bank -> squashing -> max pooling
model:add(nn.SpatialConvolutionMM(32, 64, 5, 5))
model:add(nn.Tanh())
model:add(nn.SpatialMaxPooling(2, 2, 2, 2))
-- stage 3 : standard 2-layer MLP:
model:add(nn.Reshape(64*2*2))
model:add(nn.Linear(64*2*2, 200))
model:add(nn.Tanh())
model:add(nn.Linear(200, 10))
model:add(nn.LogSoftMax())
criterion = nn.ClassNLLCriterion():cuda()
model:cuda() – To run the model on GPU
```

The above model was optimized using Stochastic Gradient Descent with a learning rate of 0.01, batch size of 100 and learningRateDecay of 5e-3.

4.3 Results and Discussions

The model took around 70 seconds to converge to a test accuracy of 98.88%. It was run for 10 epochs.

Train accuracy: 99.4%
Test accuracy: 98.88%

Test set confusion matrix and train/test loss figures are shown below:

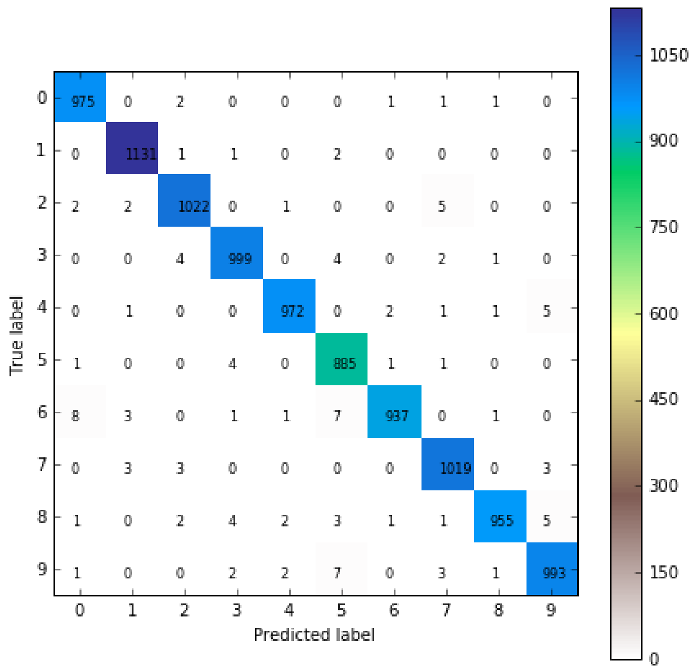


Figure 5: Test set confusion matrix for LeNet-5

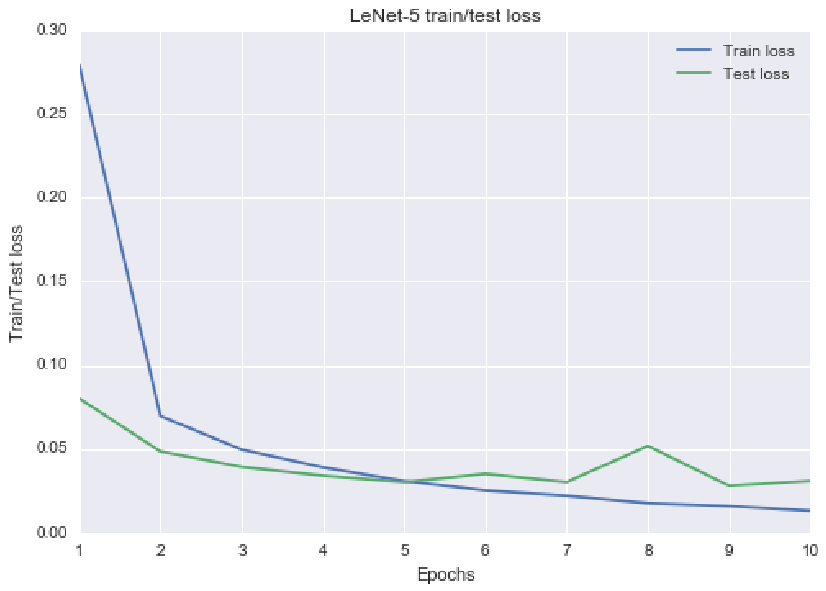


Figure 6: Train/Test loss for LeNet-5 training

5 Deep Belief Net

5.1 Experiments and Methods

This network was trained using the MATLAB code provided by Geoffrey Hinton. The training procedure is described below:

5.1.1 Training a deep belief network by stacking RBMs (Greedily)

- First train the RBM that receives input directly from raw pixels using 1-step contrastive divergence.
- Then treat the activations from first layer as they were pixels and learn the features of second hidden layer.
- For the third layer, concatenate features from second layer with image labels and train the RBM

5.1.2 Fine-tune the weights using contrastive wake-sleep algorithm

5.1.3 Fine-tune for discrimination using backpropagation

Since the generative model already has a good set of feature detectors, they can be fine-tuned for discrimination using backpropagation.

The network architecture is shown below:

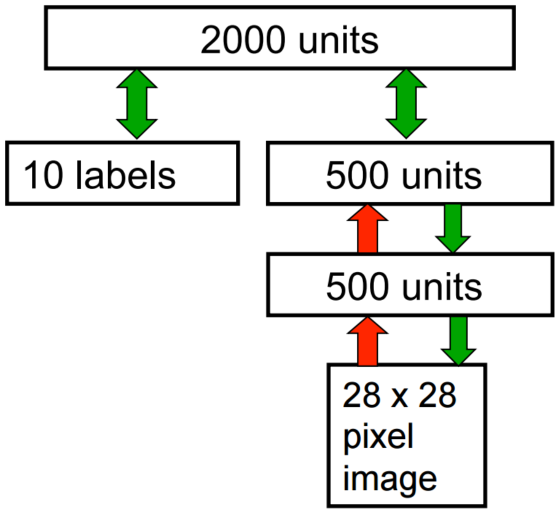


Figure 7: Deep belief net architecture

5.2 Results and discussions

The model took 6 hours to train in MATLAB using the default parameters that were specified in the code. The generative model/auto-encoder was trained for 50 epochs. Backprop was run for 200 epochs to fine-tune the discriminative model. The accuracies on train and test set are reported below:

Train accuracy: 100%
Test accuracy: 98.82%

To visualize the activations of third hidden layer using t-SNE, test images were fed to the network from bottom and the activations of third hidden layer (with 2000 neurons) were recorded for each digit. Finally, these 2000 dimensional vectors were projected to 2D space using t-SNE. The plot is shown below:

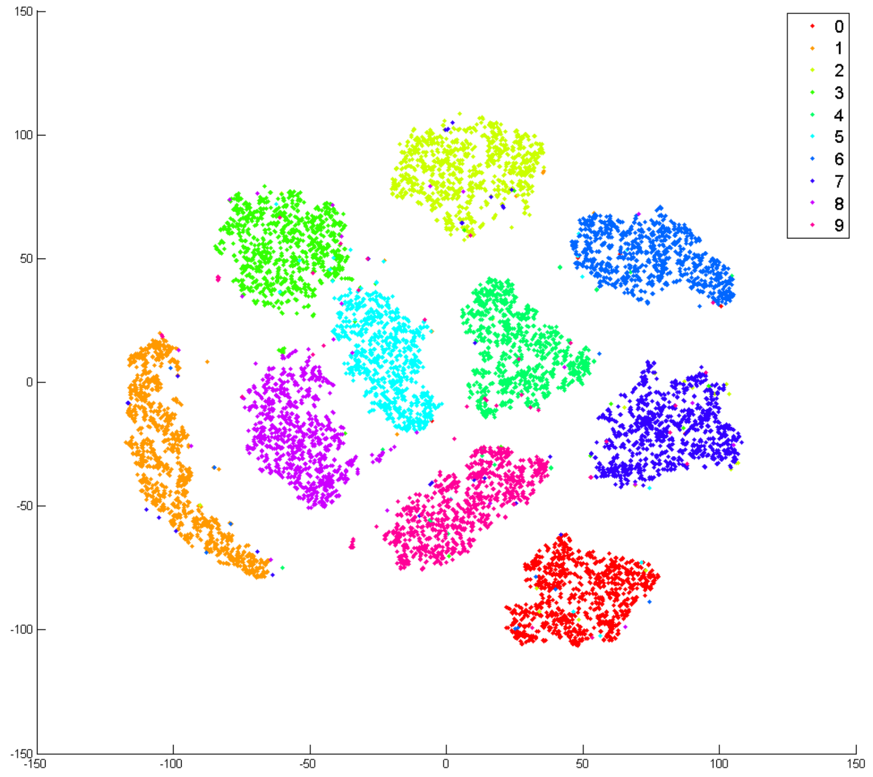


Figure 8: tSNE projection of activations of top hidden layer

As it can be seen from the activations of hidden layer, the network does a good job in recognizing different digits.

References

LeNet-5 - http://yann.lecun.com/exdb/lenet/

Torch tutorials - https://github.com/torch/tutorials

Coursera - Neural Networks for Machine Learning (Lecture 13,14) by Geoffrey Hinton

All references cited in Assignment-2 problem statement

Appendix

All the codes (iTorch/iPython notebooks, Golang files) used for this assignment are available at this GitHub repository – https://github.com/saurabh3949/UCSD-COGS-260