REPORT

Implementation details

- Network architecture:

Conv1 – RelU1 – MaxPool1 – Conv2 – RelU2 – MaxPool2 –FCL1 - RelU3 – Dropouts – FCL2 – Softmax

(FCL : fully connected layer)

- Loss function: Cross Entropy

- The code can be executed by running the following command in the terminal:

pathToPython/python *pathToFile*/main_mnist.py --data dir <directory for storing data> --nEpochs <no. of epochs> --imgNoise <stdev of noise to be added to image> --labelNoise <% of labels to randomize>

Eg: python main_mnist.py --nEpochs 5 --imgNoise 8 --labelNoise 5

The code has been adapted from the TensorFlow tutorial 'Deep MNIST for Experts'.

PART 1: Classifier on original data

Log file: logs_Part1.txt

1. Test set error rate:

After 1 iteration: 0.0377
After 5 iterations: 0.0125

2. Test set accuracies for each class:
   (After 1 iteration)

Class	Accuracy	Error rate
0	0.994898	0.005
1	0.996476	0.003
2	0.980620	0.020
3	0.981188	0.019
4	0.993890	0.007
5	0.991031	0.009
6	0.979123	0.021

7	0.983463	0.017
8	0.986653	0.013
9	0.972250	0.028

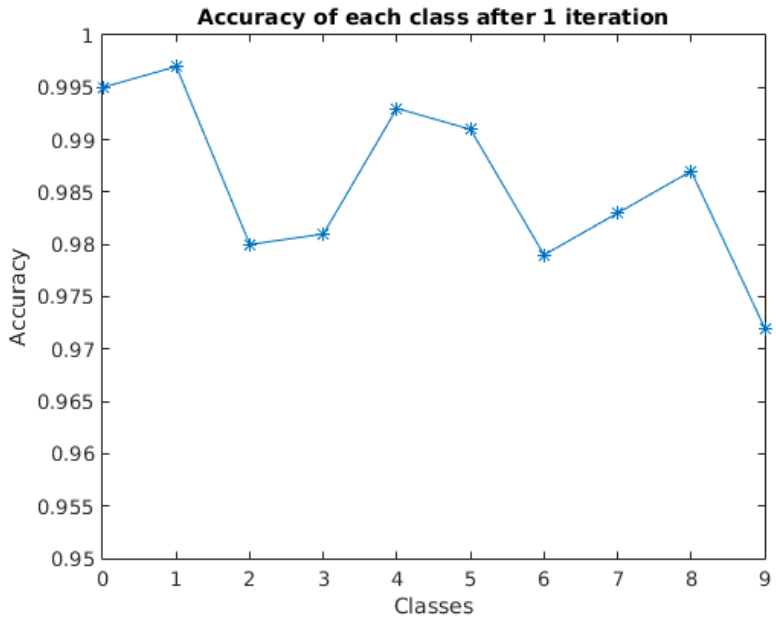

Figure 1 Class wise accuracy

Yes, some classes have higher error rates, like 2, 6, 7 and 9. There could be many reasons for this:

a. Some digits naturally have more variability in how they are written, Eg. 7 and 4. In contrast, digits like 0 and 1 achieve the best accuracies due to their relatively simple structure.
b. Numbers that differ by only 1 stroke could be confused together. For example, 2 could be misclassified as 7 and 7 as 9. The confusion matrix shows some evidence for this.

3. The training accuracy, and more clearly, the testing accuracy, shows a general trend of increasing throughout the mini batches in the 1st iteration as shown below (for a different run):

% ==== Iteration0 : Test accuracy = 0.085 ==== %

step 100, training accuracy 0.84

% ==== Iteration0 : Test accuracy = 0.8163 ==== %

step 200, training accuracy 0.98

% ==== Iteration0 : Test accuracy = 0.9062 ==== %

step 300, training accuracy 0.9

% ==== Iteration0 : Test accuracy = 0.9257 ==== %

step 400, training accuracy 1

% ==== Iteration0 : Test accuracy = 0.9398 ==== %

step 500, training accuracy 0.92

% ==== Iteration0 : Test accuracy = 0.9468 ==== %

step 600, training accuracy 1

% ==== Iteration0 : Test accuracy = 0.9493 ==== %

step 700, training accuracy 1

% ==== Iteration0 : Test accuracy = 0.9493 ==== %

step 800, training accuracy 0.92

% ==== Iteration0 : Test accuracy = 0.9586 ==== %

step 900, training accuracy 1

% ==== Iteration0 : Test accuracy = 0.9581 ==== %

step 1000, training accuracy 0.92

% ==== Iteration0 : Test accuracy = 0.9645 ==== %

step 1100, training accuracy 0.92

% ==== Iteration0 : Test accuracy = 0.9628 ==== %

% ==== Iteration1 : Test accuracy = 0.9629 ==== %

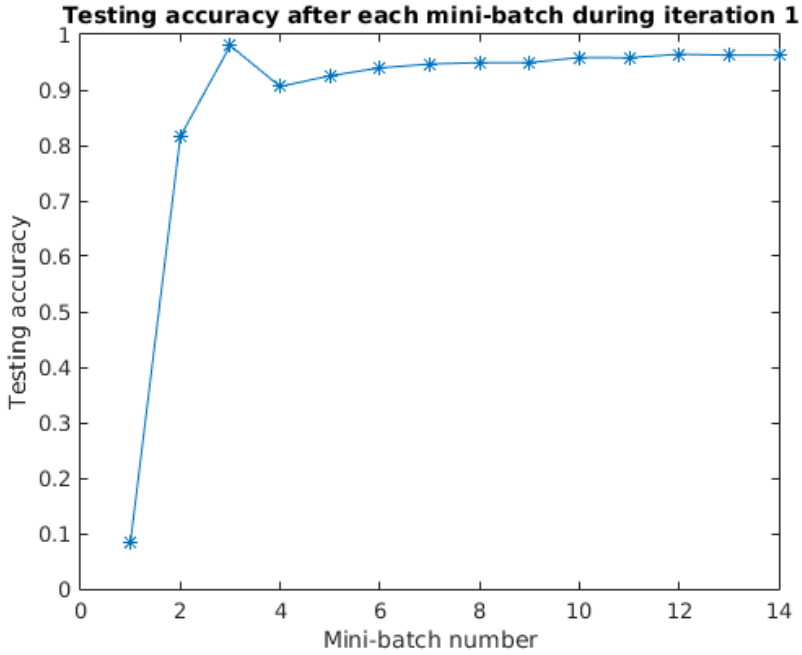

Figure 2 Testing accuracy increases continually in iteration 1

Hence the CNN might benefit with further training.

4. Further ways to improve accuracy:

a) Data augmentation

MNIST is a relatively small dataset. Though more important for larger networks than used currently in my code, data augmentation through small rotations and elastic distortions will help the network to learn more variations that occur in hand written digits and prevent over-fitting to the training dataset.

b) Regularization

Dropouts are used currently. L1/ L2 regularization is worth exploring to prevent over-fitting.

c) Deeper architectures

Currently, I use 2 convolution layers. Deeper architectures with smaller filters can be explored because as a rule of thumb, the deeper, the better, but at the expense of increased training difficulty.

d) Pre-Processing data

In contrast to (c) above, cleverly pre-processed data could allow the use of simpler networks. The images could be pre-processed in many ways, including noise removal, smoothing, de-skewing etc.

PART 2: Added image noise

Log file: logs_Part2_XX.txt

Sample image with varying levels of noise added to it:

Figure 3 Standard deviation a) 0 (No added noise) b) 8 c) 32 d) 128

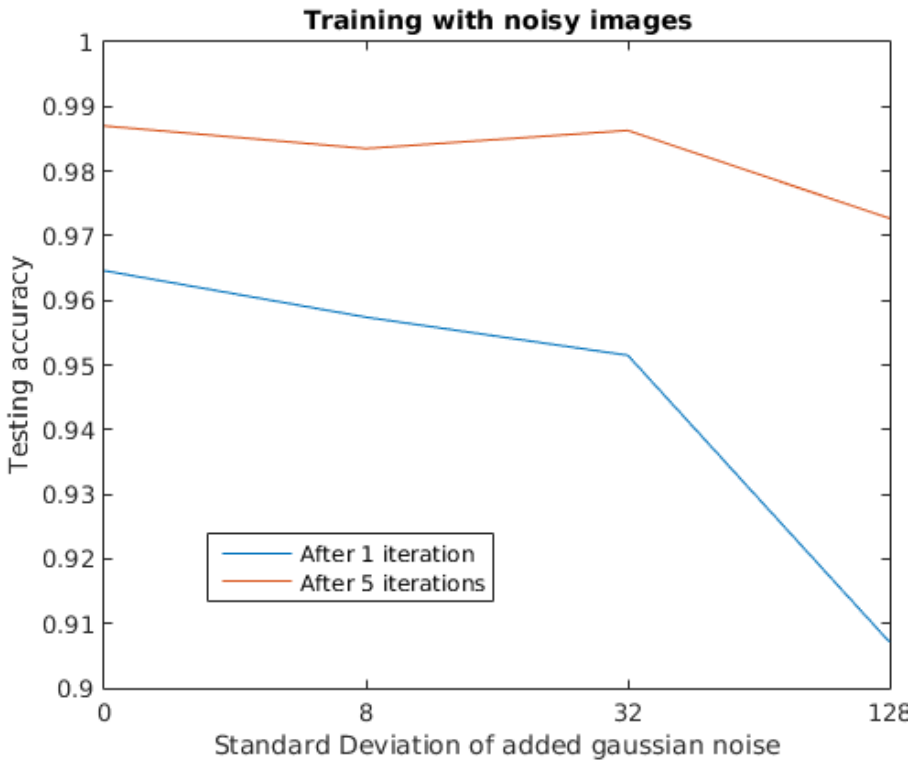

Figure 4 Testing accuracy as a function of added gaussian noise

1. Increasing image noise decreases accuracy of the classifier, especially when the image is visibly degraded as in Fig. 4 (d). This is even more important when the training images and test images have different noise levels. According to the experiments done, a production classifier could tolerate noise with standard deviation of up to 15% of the image intensity. However training for longer duration seems to compensate for this to some extent.

2. At low light situations, with long exposure times/ high sensitivity, noise in digital photographs is dominated by Shot noise which has a Poisson distribution. At normal exposures, or with low quality camera sensors, noise can be approximated as Gaussian. We can compensate for it by:

    a. Pre-processing the input images and reducing Gaussian noise
        i. Simple smoothing filters
        ii. Non local smoothing, where you average only similar pixels
        iii. Anisotropic diffusion
        iv. Statistical methods that model Gaussian noise
    b. Using deeper networks that learn features not affected by noise. However care must be taken to avoid overfitting.

c. Include noisy images (relative to test images) in the training batch. This could actually act as regularization.

3. Yes, the accuracy of classes like 2, 7 and 9 are affected more. As mentioned earlier, these have a high risk of being confused with other classes; hence noise might affect these more.

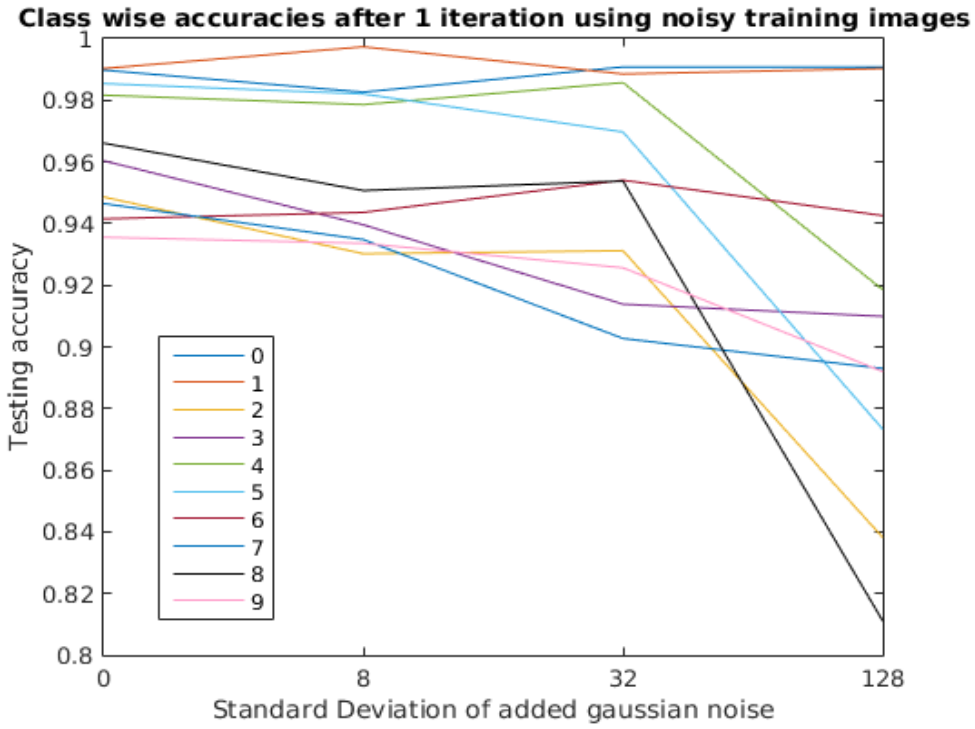

Figure 5 Class wise accuracy as a function of Gaussian noise added

PART 3: Label Noise

Log file: logs_Part3_XXpt.txt

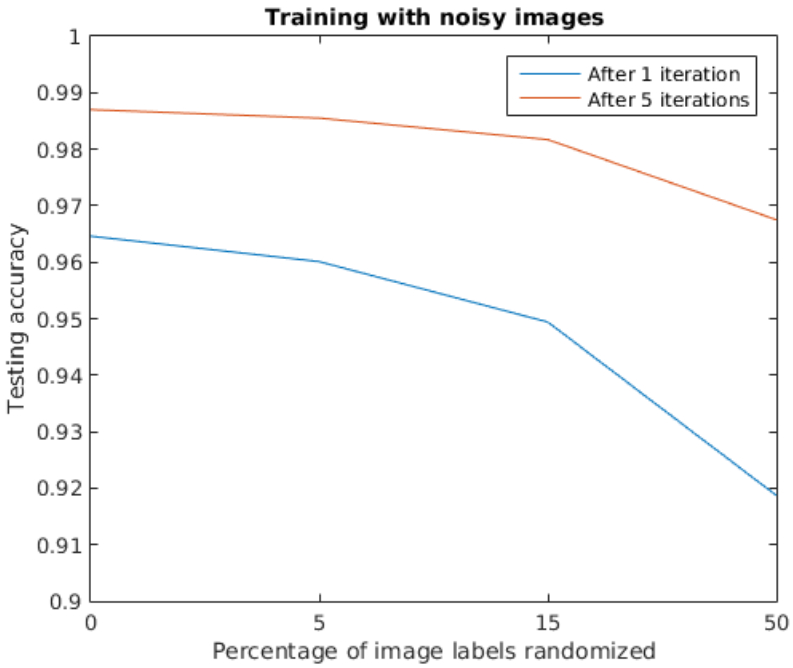

Figure 6 Testing accuracy with noisy labels

1. Accuracy goes down with increasing label noise. Training accuracy falls much faster than testing accuracy. This might be because the network does learn with the (fewer) available correctly labelled data and the testing dataset is still correctly labelled.

2.

a. Avoid overfitting to training dataset (through regularization methods like dropouts)

b. Fine tune network using only a clean subset of the large dataset

c. Detect and remove noisy labels (see que. 3 below)

d. Use loss functions robust to label noise

3. High training error inspite of long training duration could be indicative of label noise. Some ways to quantify label noise:

a. Class-wise analyses of the distribution of some simple features. (Eg: average inetsnity, morphological features etc.). Outliers could be misclassified images (or exceptions).

b. The above solution can be taken a step further by building a network to classify a single class, say '8'. Misclassified 8's would give high error during training/testing.

- Representation learning methods like autoencoders can be trained to learn an identity function to a single class, using a small clean subset of the noisy dataset. Such a network can potentially identify misclassified data when run though the whole class.

c. Use an ensemble of networks. If a large proportion of the networks misclassify a particular instance, it is probably misclassified.

d. A rough initial estimate could be obtained by picking a random subset of images and manually checking the labels.

4. I would be concerned about label noise as it is comparatively more difficult to detect in large datasets and compensate for, as compared to image noise. This is based on the assumption that the images that the network sees have noise levels similar to real world images, else the network will perform poorly, as shown in PART 2. Noisy training images, representative of real world images, are preferable to noisy labels. However high image noise might make it difficult for the network to learn anything at all.