**Report of Deep Learning Capstone Project**

Xuanzhe Wang

**Definition**

In this project, I trained a convolutional neural network to classify real world street digits.

The data set consists of 73257 training images and 26032 testing images. Each image are 32x32 bitmap(array) with 3 color channels.

**Methodology**

**1. Data Preprocessing**

Each pixel in the image has a range from 0 to 255. In order to let gradient descent work, first I normalized then into -1.0 to 1.0 as 32-bit float. This step was forgotten at first, and I got billions of losses in the training phase.

Let’s have a visual impression between several original and normalized images.

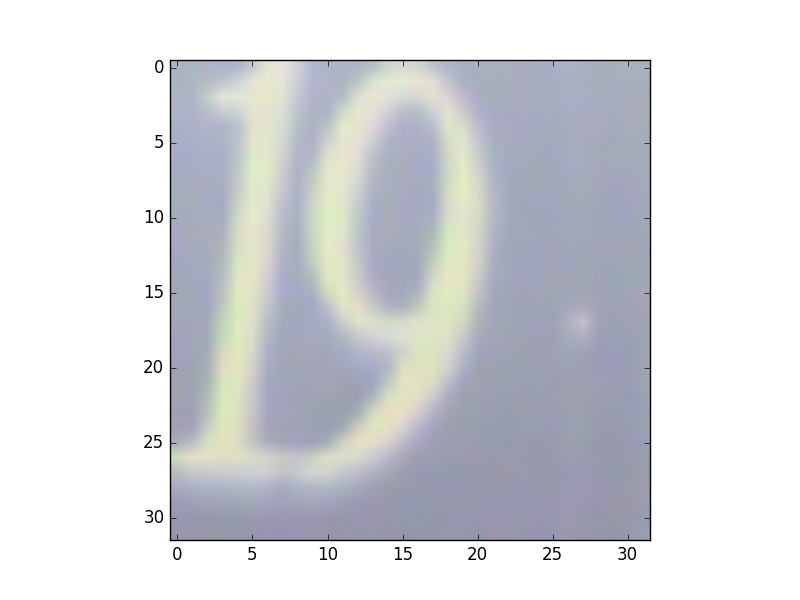


Figure 1, image\_1\_origin, label 9

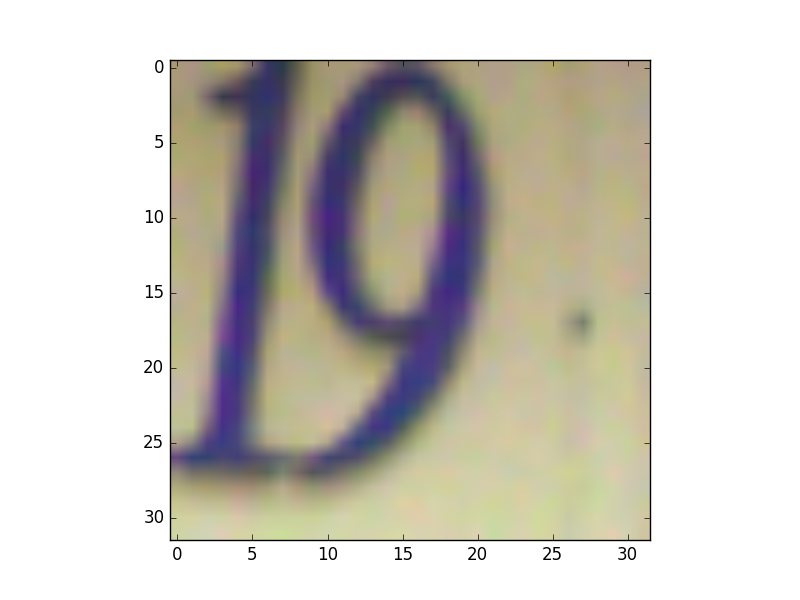


Figure 2, image\_1\_normal, label 9

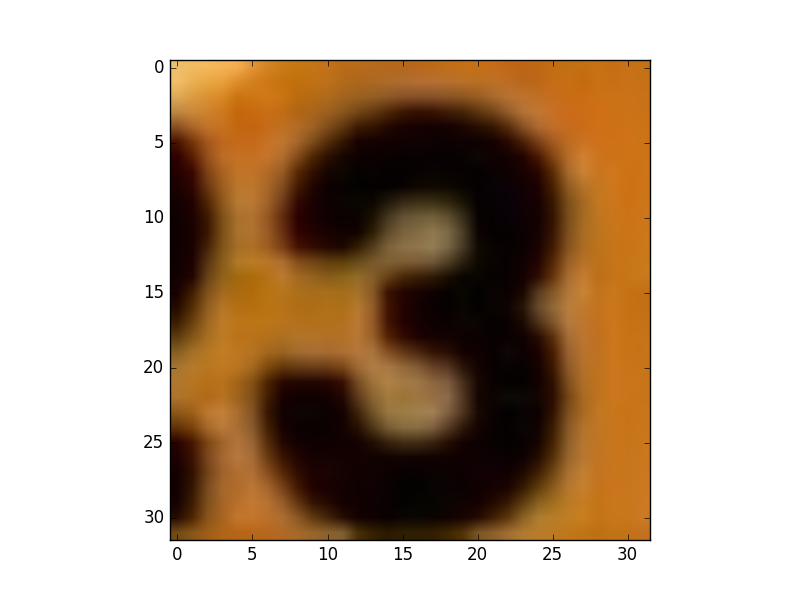


Figure 3, image\_2\_origin, label 3

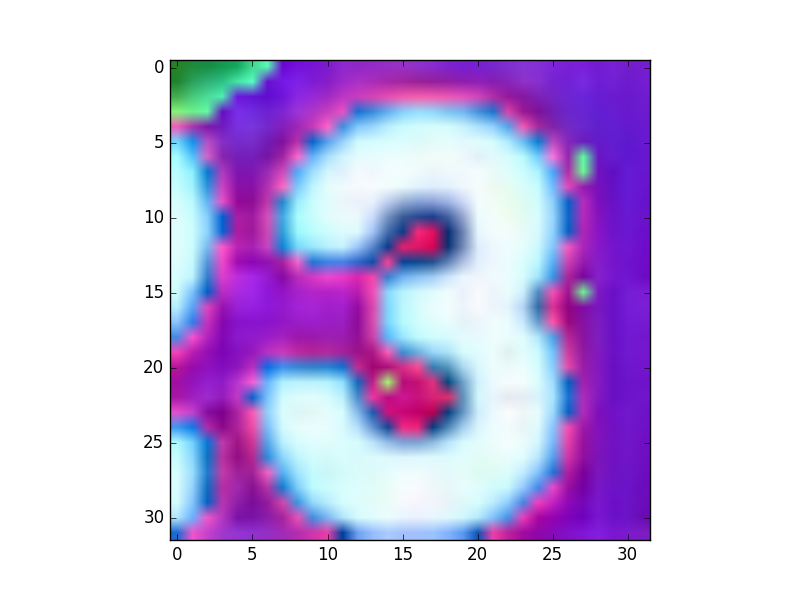


Figure 4, image\_2\_normal, label 3

As we can see, the normalized images have a stronger contrast. But this virtual impression should not be considered as numeric evidence since images were drawn by matplotlib, the python library. The point is that by applying a linear mapping from [0 ~ 255] to [-1.0 ~ 1.0], we obtain the same information but have nicer floating point to work with. I will explain the meaning of “nicer” later.

Another observation is that some images contains multiple digits. Right labels for such images are the digits closer to the center.

**2. Three Graph Architectures**

1) 2 Convolution + 2 Fully Connected

Conv -> Relu -> Max Pool -> Conv -> Reul -> Max Pool -> Dropout -> Fully Connected -> Fully Connected

2) 3 Convolution + 2 Fully Connected

Conv -> Relu -> Conv -> Relu -> Max Pool -> Conv -> Reul -> Max Pool -> Dropout -> Fully Connected -> Fully Connected

3) 4 Convolution + 2 Fully Connected

Conv -> Relu -> Conv -> Relu -> Max Pool -> Conv -> Reul -> Conv -> Relu -> Max Pool -> Dropout -> Fully Connected -> Fully Connected

Due to limited computing power, I cannot vertically scale deeper. One interesting observation is that deeper network doesn’t improve the result at all, possibly because 2 convolutional layers are enough to catch all the information in the input since the input are not complex. I will expand the discussion more in **Analysis** section.

Hyper Parameters are below:

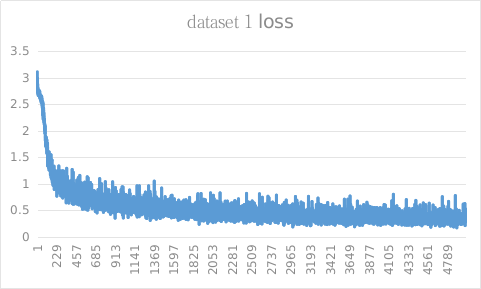
1. Number of hidden nodes in fully connected layer
2. Depth of each convolutional layers
3. Patch size
4. Pooling stride
5. Drop out rate
6. Number of iterations to train
7. Base learning rate assuming exponential decay
8. Decay rate

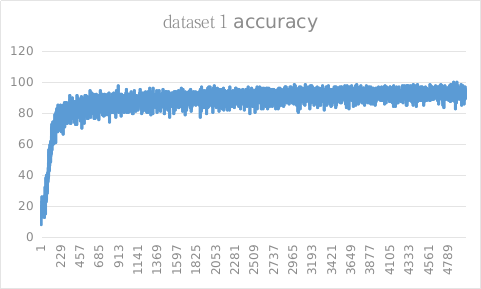
Number of iteration is not a hyper parameter of the network, but just a variable I use to control the training time. Batch size and number of iteration are complements. Basically I just set the batch size as big as memory capacity.

**Analysis**

Through a lot of experiments and parameter tuning, I discovered that number of hidden nodes, convolutional depth have almost no influence. Patch\_size, and drop\_out\_rate have more influence on loss and accuracy. Additionally, the choice of optimizer function, base learning rate and decay rate are vital to the result.

Let’s look at training data.

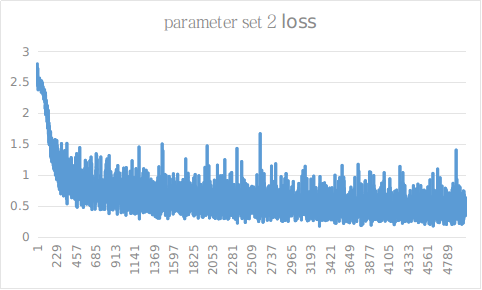


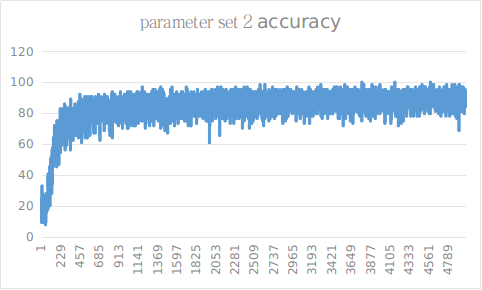
****

1. num\_hidden=128
2. batch\_size=128
3. patch\_size=5
4. conv1\_depth=32
5. conv2\_depth=32
6. pooling\_stride=2
7. drop\_out\_rate=0.9
8. base\_learning\_rate=0.0013
9. decay\_rate=0.99
10. optimizer='adam'

This is the optimal configuration I have get so far. The loss converges to ~0.3 and the accuracy converges to ~93%.

**Observation 1: Number of hidden nodes of fully connected layer has almost no influence.**





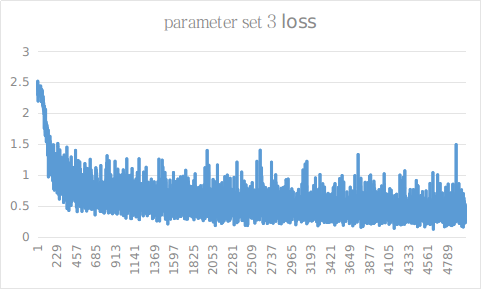
1. num\_hidden=64
2. batch\_size=64
3. patch\_size=5
4. conv1\_depth=32
5. conv2\_depth=32
6. pooling\_stride=2
7. drop\_out\_rate=0.9
8. base\_learning\_rate=0.0013
9. decay\_rate=0.99
10. optimizer='adam'

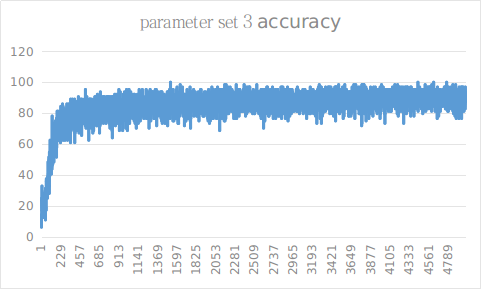
Here I changed num\_hidden from 128 to 64. I also changed batch\_size from 128 to 64, but this is purely for faster training.

We can see that the average loss and accuracy converges to almost the same value. The speed of convergence is also similar. Both parameter set 1 and parameter set 2 converges around 1000 iterations. One big difference is that both loss and accuracy oscillates much more, comparing to parameter set 1. This indicated that the convergence has a higher deviation.

My explanation is that a lower number of hidden layer nodes in the fully connected layer produce less final features to the softmax function, which gives softmax function more “pressure” to do the final decision(classification). However, because my convolutional layers have done a good feature extraction job, less final features in the fully connected layer is not a huge problem, as long as number of hidden nodes is not too small.

**Observation 2: Convolutional depth has almost no influence.**

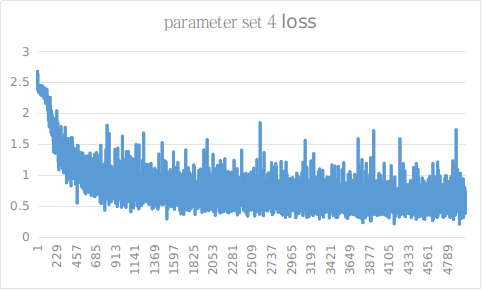
****

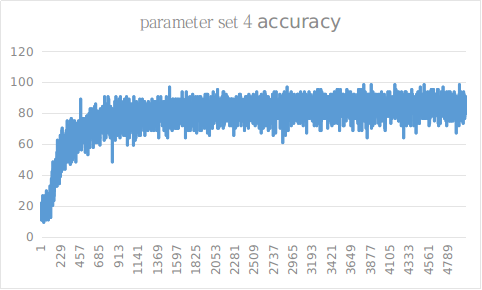
****

1. num\_hidden=64
2. batch\_size=64
3. patch\_size=5
4. conv1\_depth=16
5. conv2\_depth=16
6. pooling\_stride=2
7. drop\_out\_rate=0.9
8. base\_learning\_rate=0.0013
9. decay\_rate=0.99
10. optimizer='adam'

Here I changed convolutional layers’ depth from 32 to 16. However, the result is almost identical to parameter set 2’s. Possibly, because the problem domain is very narrow in turns of that inputs are just digit numbers, there are not many features to extra from the input, therefore, 16 depth is enough for this problem.

**Observation 3: Drop out rate is essential for a good model.**

****

****

1. num\_hidden=64
2. batch\_size=64
3. patch\_size=5
4. conv1\_depth=16
5. conv2\_depth=16
6. pooling\_stride=2
7. drop\_out\_rate=0.5
8. base\_learning\_rate=0.0013
9. decay\_rate=0.99
10. optimizer='adam'

Here I changed drop out rate from 0.9 to 0.5. The loss converges to a higher value around 0.7 and the accuracy drops to around 80%. The speed of convergence becomes slower because we can see a more curve shape at the beginning.

One intuition is that drop out acts like a weak learner ensemble method in turns of that the final fully connected layer only gets the input from a subset of the whole network.

With a 0.5 dropout rate, half of the signal from the first fully connected layer to be randomly dropped. Therefore, at each training iteration, the second fully connected layer, which is the last layer in front of softmax funciton, can not rely on the inputs from fc1 totally.

This effect is like to ask fc2 to make a decision based on less features. Additionally, each iteration will only have half gradients been back propagated through the network. The network learns more conservatively in some sense.

With a higher drop out rate such as 0.9, the network is much more conservatively. I am not saying that a high dropout rate is always better. But for the objectives of our problem, it is.

**Results**

**Conclusion**