**Deep Learning Capstone Report**

Xuanzhe Wang

**Table of Content**

**I. Definition**

1. **Project Overview**
2. **Problem Statement**
3. **Metrics**

**II. Analysis**

1. **Data Exploration**
2. **Exploratory Visualization**
3. **Algorithms and Techniques**
4. **Benchmark**

**III. Methodology**

1. **Data Preprocessing**
2. **Implementation**
3. **Refinement**

**IV. Result**

1. **Model Evaluation and Validation**
2. **Justification**

**V. Conclusion**

1. **Free-Form Visualization**
2. **Reflection**
3. **Improvement**

**I. Definition**

a. Project Overview

Project is to solve an image classification problem. In machine learning, classification problem is to categorize inputs. For example, given several images, the machine learning agent need to tell which image is a car, which image is a person.

That is, given an input X (X is a vector), we need to find a function f that:

f(x) -> y, where y is the category/label of x

In this project, a special technical called convolutional neural network is applied. Convolutional neural network has been applied in image recognition, video analysis, drug discovery and other fields. For example, the famous AlphaGo by DeepMind used convolutional neural network.

b. Problem Statement

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

The data set used is the [Street View House Numbers (SVHN): A large-scale dataset of house numbers in Google Street View images.](http://ufldl.stanford.edu/housenumbers/)

The data set consists of 73257 training images and 26032 testing images. Each image is a 32x32 bitmap(array) with 3 color channels. Each image also has a label associated with it. These labels represent digits from 0 - 9.

My task is to train a model, as a function f, such that:

f(Image) -> label, where label ranges from 0 - 9

c. Metrics

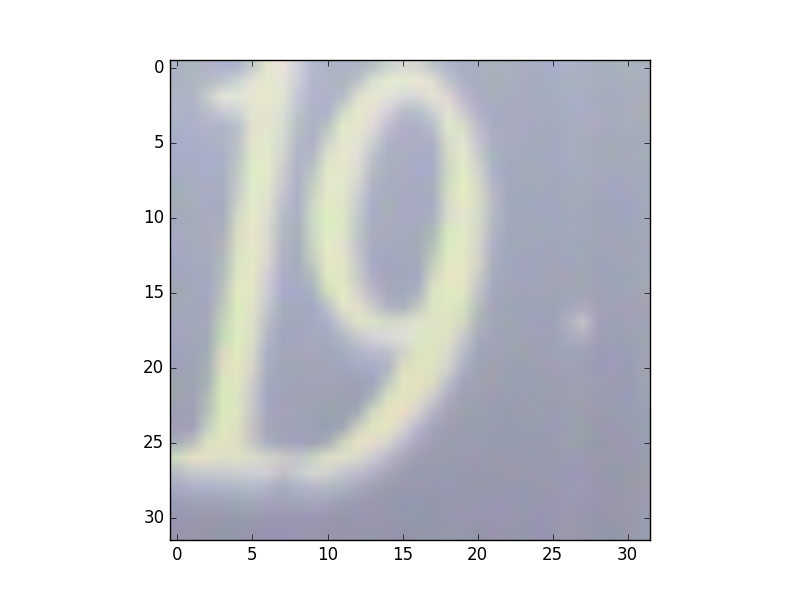
Accuracy: the number of correctly classified inputs / the number of inputs

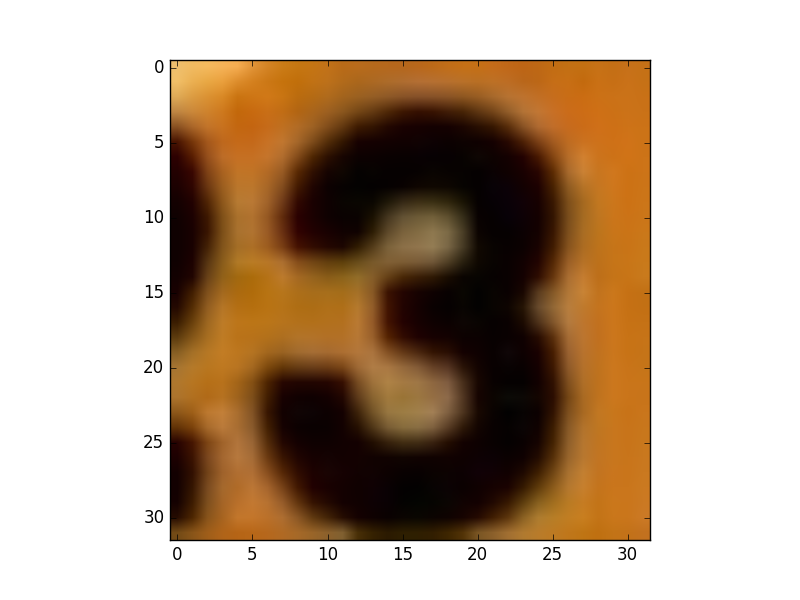
**II. Analysis**

1. Data Exploration

There are 73257 images in the training set and 26032 images in the testing set. Each image is represented as a 3-channel RGB 32x32 array.

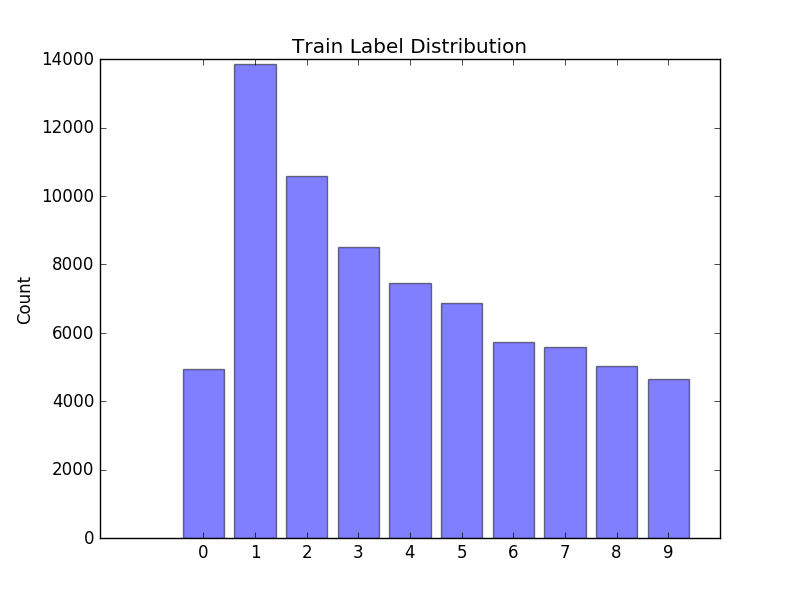
1. Exploratory Visualization

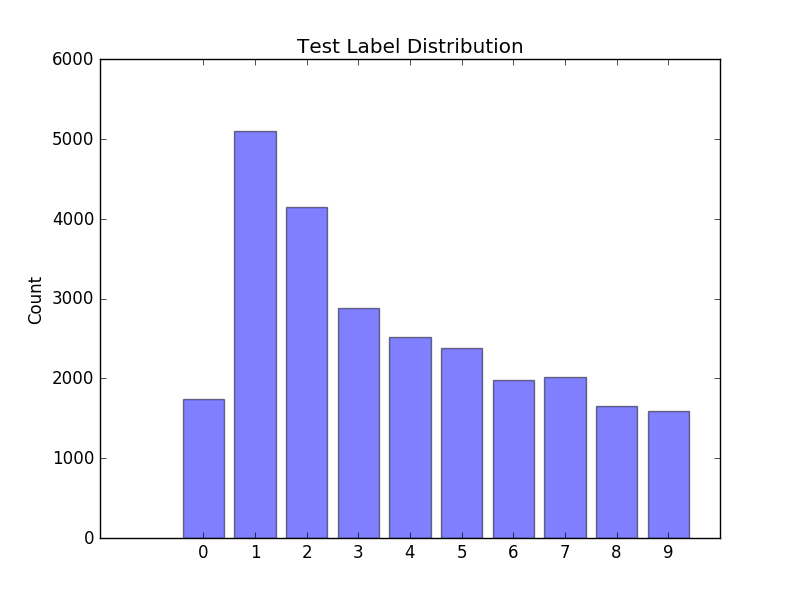
Figure 1, image\_1\_origin, label 9

Figure 3, image\_2\_origin, label 3

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

The image below is the distribution of train labels and test labels. As you can see, distributions in the training set and the testing set are similar.





1. Algorithms and Techniques

I applied three different architectures to the problem.

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

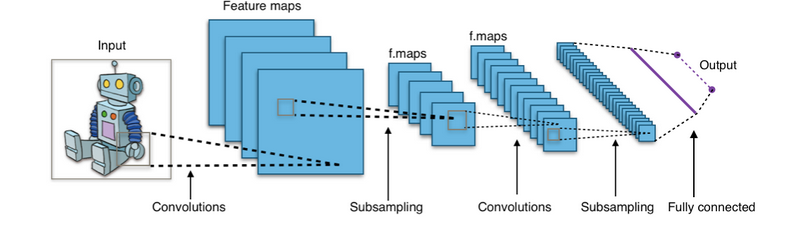
3) 4 Convolution + 2 Fully Connected

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

**Explanations of Different Layers:**

**Convolutional Layer:**

Given an image with size N x N. A convolution is to slice through the image with a window size M x M, where M < N. In each slice, compute the matrix multiplication of the M x M cut of the image and a M x M weight matrix. All the outputs will form a new matrix and send to the next layer as the input.



One reason that convolution is good for image problems is that convolution preserves the local information of a image by the slicing window operation.

**Relu:**

A rectifier function function(X) = max(0, X)

This is a thresholding function that outputs all the positive inputs. If inputs are non-positive, outputs 0.

**Max Pooling:**

Max Pooling is just an image compression/down sizing step. If the stride is 2, then it means to downsize the image by a factor of 2. For example:

We have a single channel image

12, 34, 45, 03

08, 10, 52, 27

82, 99, 00, 43

66, 21, 02, 91

The max pooling function will only get the max value of every 2x2 cut of the original matrix. The output will be

34, 52

99, 91

**Dropout:**

Given a vector input X, randomly overwrite some values in X as 0 at a certain dropout rate. For example, if the dropout rate is 0.6:

<1, 3, 5, 7, 9> = dropout => <0, 3, 0, 0, 9>

**Fully Connected Layer:**

Normal neural network layer

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.

1. Benchmark

I started with the architecture number 1 mentioned in the Methodology section.

The initial parameter setting is:

Number of hidden nodes = 64

Batch\_size = 128

Patch\_size = 5

Convolutional layer depth = 16

Pooling Stride = 2

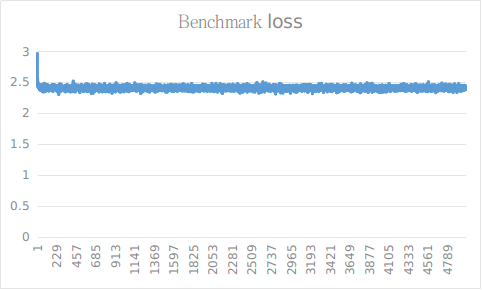
Dropout rate = 0.5

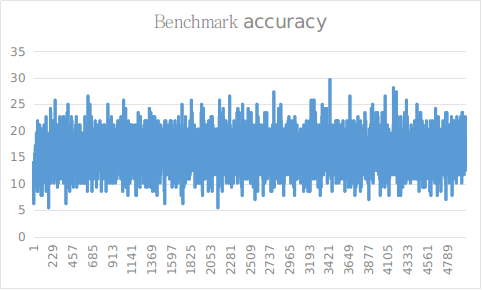
Base learning rate = 0.05

Decay rate = 0.95

Gradient Descent Optimizer

And the results are below:





As we can see, the model is useless. It converges prematurely.

The loss converges to 2.5 and the accuracy is 15% averagely.

**III. Methodology**

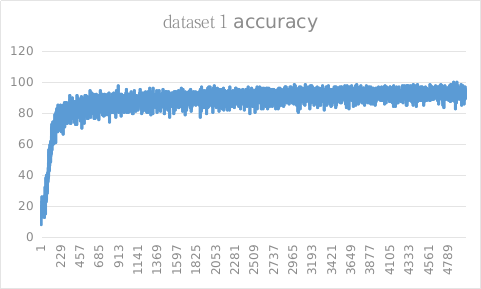
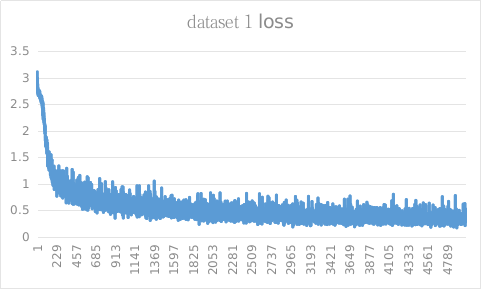
1. Data Preprocessing

Each pixel in the image has a range from 0 to 255. In order to let gradient descent work, first I normalized them 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 because gradient descent can’t update big numbers effectively.

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 also grayscaled all images to get a better memory performance. The results of grayscaled images and 3-channel images are very similar. Therefore grayscaling them is a wise choice.

1. Implementation
2. Refinement

Through a lot of experiments and parameter tuning, I discovered that number of hidden nodes and convolutional depth have almost no influence. Patch size and dropout 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. 

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%.

**IV. Result**

1. **Model Evaluation and Validation**
2. **Justification**

**V. Conclusion**

1. **Free-Form Visualization**
2. **Reflection**
3. **Improvement**