

# Capstone Project

Machine Learning Engineer Nanodegree

Sungju Kwon  
December 12st, 2016

## Definition

### Project Overview

Single digit recognition problem like MNist is most common example for machine learning algorithm. Multi digit recognition is more challenging problem. In this project, I'll implement convolutional neural network that recognize multi digits and upload that model to Android application based on Tensorflow Android demo application<sup>1</sup>.

This project based on Goodfellow et al. (2014)<sup>2</sup> and Tensorflow CIFAR-10 tutorial model<sup>3</sup>. First, I convert Tensorflow CIFAR-10 tutorial model can accept SVHN dataset<sup>4</sup> and do multi digit recognition. And then tuning hyperparameters for more accuracy. And then I change Tensorflow Android camera example application can processing with my new neural network graph. Finally, I upload my neural network graph to this Android application.

Implementing Android application that can processing with Tensorflow neural network is so exciting task because it can enhanced to many other subjects(Artificial Intelligence, Self-Driving Car, Google Tango<sup>5</sup>, etc). And I can receive feedback for my model's shortage immediately. So I decide my model upload to Android application.

Goodfellow's research shows that Convolutional network's accuracy can reach to over 95%. But my goal of this project is 90% accuracy and upload that model to Android app.

### Problem Statement

Main goal is make multi digit classifier using neural network and Android application that can recognize multi digits. Goodfellow's paper shows that without complex processing, neural network classifier can do well in multi digit classifying task. So I decide follow that approach.

The tasks involved are the following:

1. Preprocessing SVHN data.
  - a. Load SVHN dataset using fuel<sup>6</sup>.

---

<sup>1</sup> <https://github.com/tensorflow/tensorflow/tree/r0.11/tensorflow/examples/android>

<sup>2</sup> Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S., and Shet, V. Multi-digit number recognition from street view imagery using deep convolutional neural networks. In Proc. ICLR, 2014.

<sup>3</sup> [https://www.tensorflow.org/versions/r0.11/tutorials/deep\\_cnn/index.html#cifar-10-model](https://www.tensorflow.org/versions/r0.11/tutorials/deep_cnn/index.html#cifar-10-model)

<sup>4</sup> <http://ufldl.stanford.edu/housenumbers/>

<sup>5</sup> <https://get.google.com/tango/>

<sup>6</sup> <https://fuel.readthedocs.io/en/latest/api/dataset.html#module-fuel.datasets.svhn>

- b. Cropped digits image by minimum boundary.
    - c. Labeling each digit and length.
      - i. 0-9 as number.
    - d. Pickling data for later use.
  2. Make multi digit classifier.
    - a. Convert Tensorflow CIFAR-10 tutorial model.
      - i. Change model can accept SVHN dataset as input.
      - ii. Add digit length classifying output.
      - iii. Add each digit classifying outputs.
    - b. Train model.
      - i. Tuning hyperparameters.
      - ii. Check accuracy like Goodfellow's model. Partial correct classifying regard as wrong.
  3. Make Android camera application
    - a. Convert Tensorflow Android camera example application.
      - i. Upload my multi digit classifier to Android application.
      - ii. Change input/output interface from Android application to tensorflow graph.
      - iii. Change Android application's displaying view. Print each digit/length result.
  4. Show real world result.

## Metrics

I define accuracy as whole digits is correct. According to Goodfellow et al. (2014), partial recognition has "no credit". I follow that policy. Accuracy is

$$accuracy = \frac{fully\ correct\ counts}{total\ counts}$$

We can think about other metric, like F1-score. But SVHN's classifying labels are very various(0 to 99999). Cheating is impossible by return specific label only. So I think accuracy is enough metric to this problem.

I'll check Minibatch/validation/test accuracy for SVHN dataset. And get some real world prediction result by Android application.

## Analysis

## Data Exploration

SVHN is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting.<sup>7</sup>

- 10 classes, 1 for each digit. Digit '1' has label 1, '9' has label 9 and '0' has label 10.
- 3 channels(RBF channels).
- 73,257 digits for training, 26,032 digits for testing, and 531,131 additional, somewhat less difficult samples, to use as extra training data
- Comes in two formats:
  - Type 1. Original images with character level bounding boxes.
  - Type 2. MNIST-like 32-by-32 images centered around a single character (many of the images do contain some distractors at the sides).

SVHN provides images and mat files(Matlab file). But i'm not familiar deal with that file in python. So, i use fuel library for load that dataset. It reads whole dataset(train, test, extra) and return one large dataset as result. Fuel's result dataset size is 248,822.

For each data in fuel's result dataset has 6 column. Each column's data is like following:

1. Column 0 : image height.
2. Column 1 : labels.
3. Column 2 : image left boundary.
4. Column 3 : image top boundary.
5. Column 4 : image width.
6. Column 5 : image byte array.

Also, I use some real world data by Android application. I collect these data by using Android application itself. Tensorflow Android camera example can feed Android camera's live input. So, I just capture some images in surroundings. 16 images are captured, and that images are saved in github repository<sup>8</sup>. 13 for 2 digits, 1 for 3 digits, 2 for 4 digits.

---

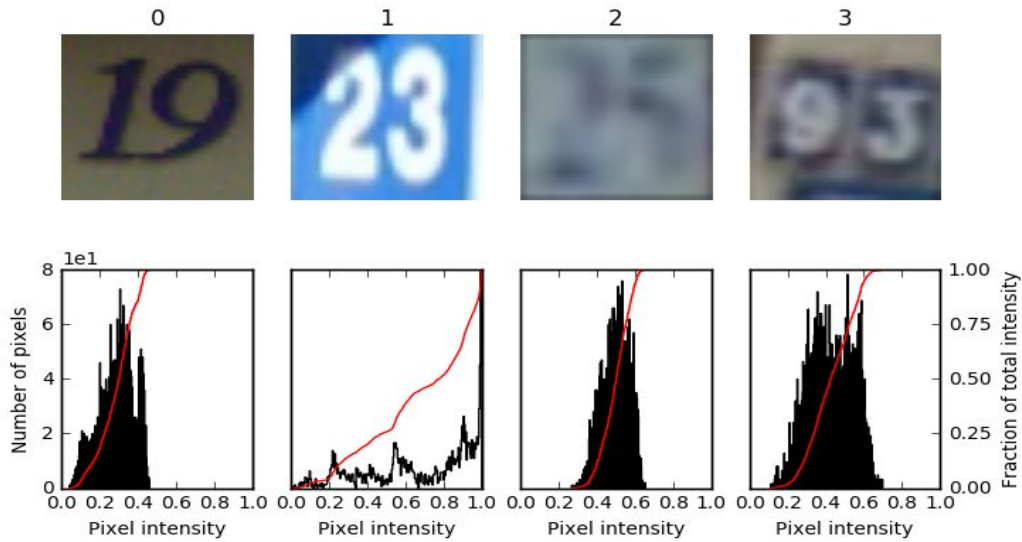
<sup>7</sup> <http://ufldl.stanford.edu/housenumbers/>

<sup>8</sup>

[https://github.com/voyageth/udacity-machine-learning-projects/tree/master/projects/capstone/deeplearning/real\\_world\\_data](https://github.com/voyageth/udacity-machine-learning-projects/tree/master/projects/capstone/deeplearning/real_world_data)

## Exploratory Visualization

**Fig. 1** SVHN image examples with pixel intensity. Some images are too many blurred or out of focussed. So preprocessing blurring can worse classifying result.



**Fig. 2** SVHN digit count histogram. Most images digit length is 3 or 4. Some image has length 6. I'll ignore length 6 images because number of data is so small(126/248,822). Digit length 2, 5 is also small but I think it is not significantly different(digit length 2 data count : digit length 6 data count = 1:6).

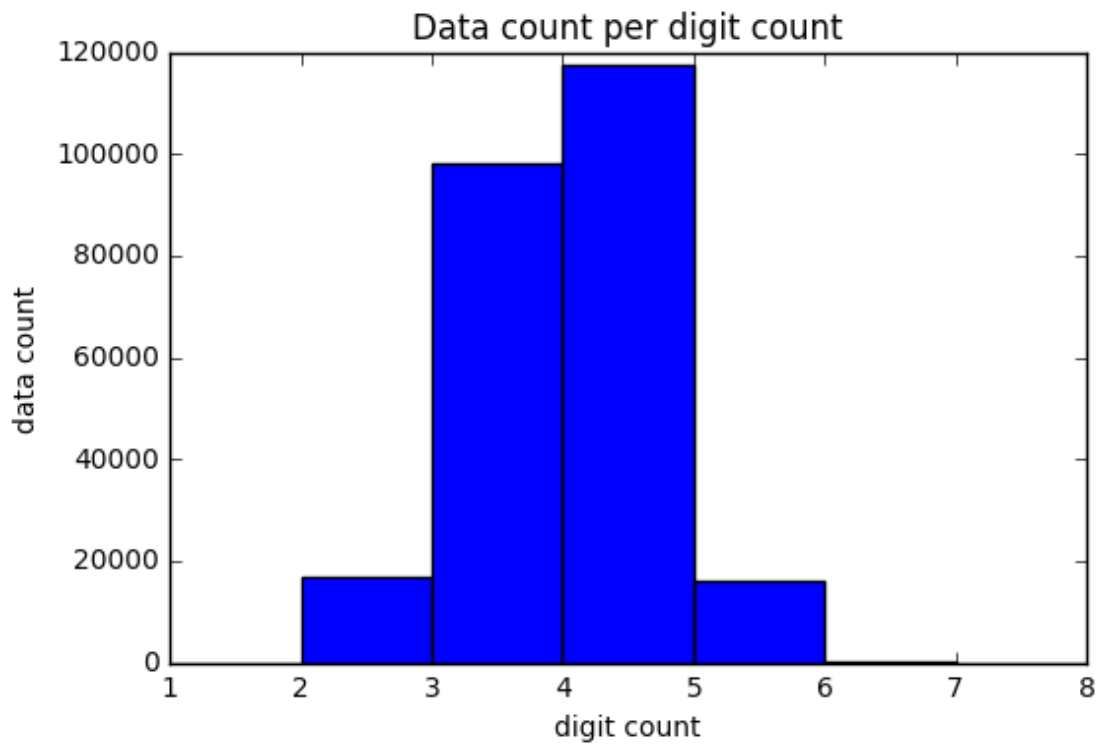
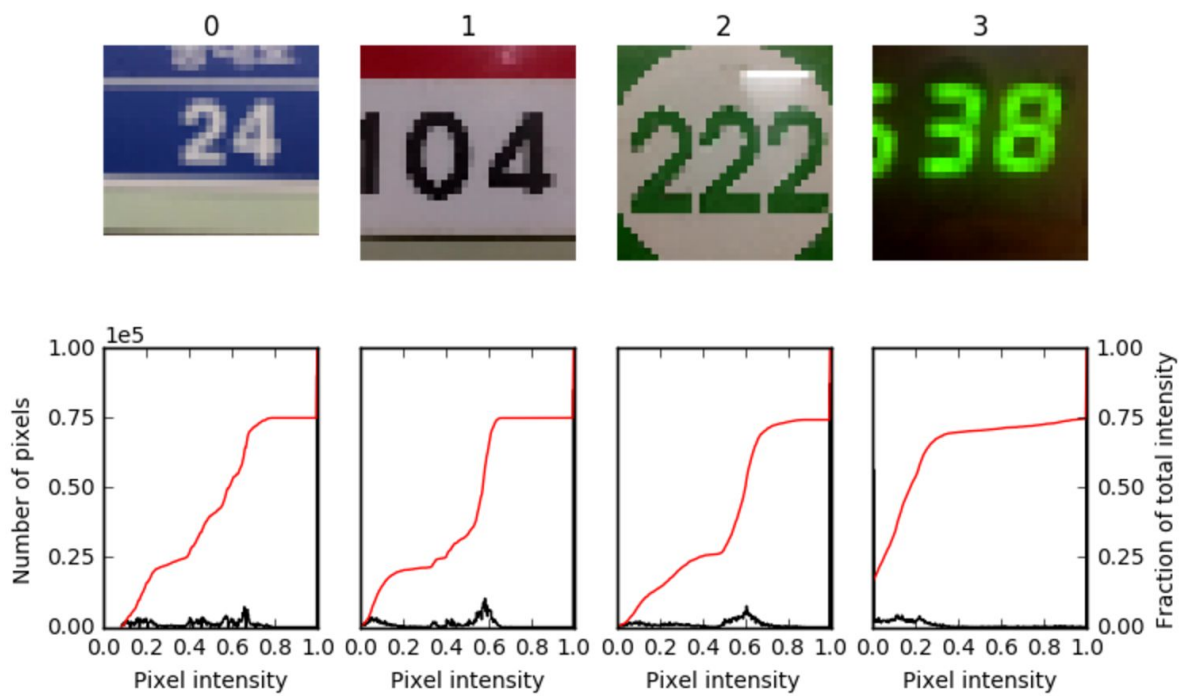


Fig. 3 Real world images.



## Algorithms and Techniques

This Project based on Tensorflow CIFAR-10 tutorial model. It use Convolutional Neural Network.

### Neural Network<sup>9</sup>

Neural networks (also referred to as connectionist systems) are a computational approach which is based on a large collection of neural units loosely modeling the way a biological brain solves problems with large clusters of biological neurons connected by axons.

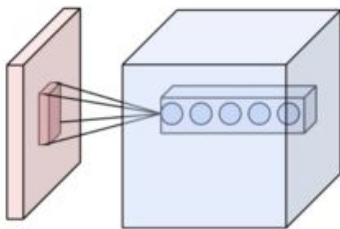
### Convolutional Neural Network<sup>10</sup>

In machine learning, a convolutional neural network (CNN, or ConvNet) is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex. Individual cortical neurons respond to stimuli in a restricted region of space known as the receptive field.

### Convolution<sup>11</sup>

The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.

**Fig. 4** Neurons of a convolutional layer, connected to their receptive field



### Max Pooling<sup>12</sup>

Max pooling is a sample-based discretization process. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned.

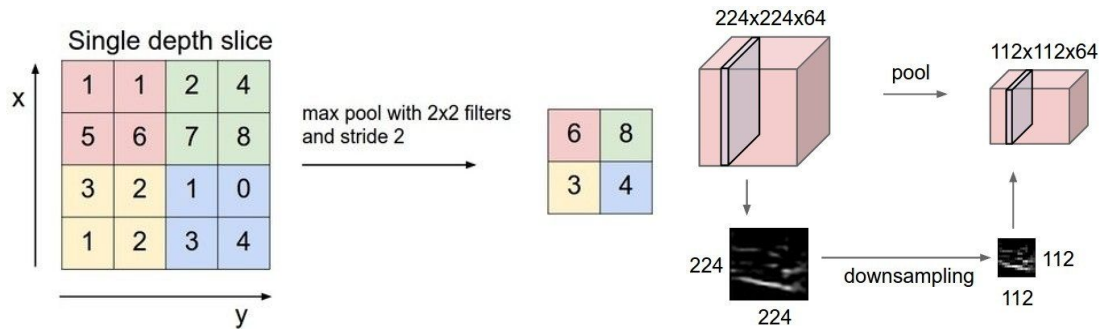
**Fig. 5** Max Pooling

<sup>9</sup> [https://en.wikipedia.org/wiki/Artificial\\_neural\\_network](https://en.wikipedia.org/wiki/Artificial_neural_network)

<sup>10</sup> [https://en.wikipedia.org/wiki/Convolutional\\_neural\\_network](https://en.wikipedia.org/wiki/Convolutional_neural_network)

<sup>11</sup> [https://en.wikipedia.org/wiki/Convolutional\\_neural\\_network#Convolutional\\_layer](https://en.wikipedia.org/wiki/Convolutional_neural_network#Convolutional_layer)

<sup>12</sup> <https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks>



### Stochastic gradient descent Optimizer<sup>13</sup>

Stochastic gradient descent (often shortened to SGD), also known as incremental gradient descent, is a stochastic approximation of the gradient descent optimization method for minimizing an objective function that is written as a sum of differentiable functions. In other words, SGD tries to find minimums or maximums by iteration.

### Adagrad Optimizer<sup>14</sup>

Adagrad is a theoretically sound method for learning rate adaptation which has the advantage of being particularly simple to implement. The learning rate is adapted component-wise, and is given by the square root of sum of squares of the historical, component-wise gradient.

### Learning rate<sup>15</sup>

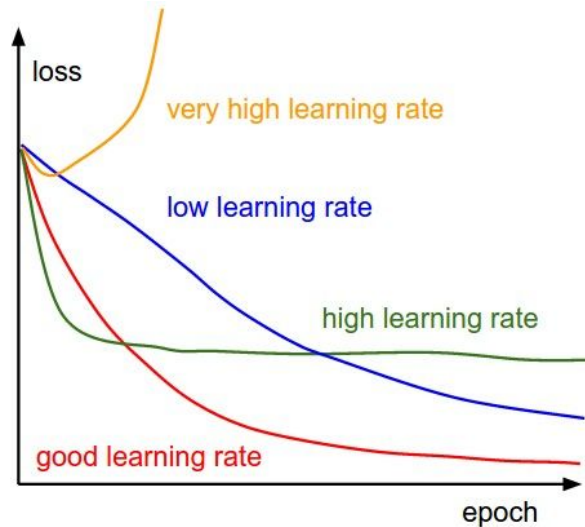
With low learning rates the improvements will be linear. With high learning rates they will start to look more exponential. Higher learning rates will decay the loss faster, but they get stuck at worse values of loss (green line). This is because there is too much "energy" in the optimization and the parameters are bouncing around chaotically, unable to settle in a nice spot in the optimization landscape.

**Fig. 6** Learning rate. epoch vs. loss

<sup>13</sup> [https://en.wikipedia.org/wiki/Stochastic\\_gradient\\_descent](https://en.wikipedia.org/wiki/Stochastic_gradient_descent)

<sup>14</sup> <https://xcorr.net/2014/01/23/adagrad-eliminating-learning-rates-in-stochastic-gradient-descent/>

<sup>15</sup> <http://cs231n.github.io/neural-networks-3/>



### Batch size<sup>16</sup>

The batch size will affect how 'noisy' the gradient is.

Neural Network has lots of hyperparameters. So we can try various parameter sets.

Most parameters are use Tensorflow CIFAR-10 tutorial model's default value. Because i think it is fine tuned for image recognition. So, i experiment following parameters.

- Convolutional layer count : 2 to 8
- Each convolutional layer node count : various
- Each fully connected layer node count : various
- Optimizer : GradientDescentOptimizer, AdagradOptimizer
- Image Size : 32, 52
- Number of steps : 10001, 100001, 200001
- Batch size : 32, 64, 128

### Benchmark

Convert Tensorflow CIFAR-10 tutorial model can accept SVHN input and can predict digits and convert optimizer to AdagradOptimizer, I gain Minibatch accuracy 79.7%. I regard that as base model. Goodfellow's paper shows that Convolutional Neural Network can predict over 95% accuracy. But due to my experiment machine's performance, I decide my goal as 90% accuracy. It increase about 10% from base model(Tensorflow CIFAR-10 tutorial model).

---

<sup>16</sup>

<https://www.kaggle.com/c/datasciencebowl/forums/t/12634/can-someone-explain-what-batch-size-is-doing-in-convolutional-nns>



# Methodology

## Data Preprocessing

Load SVHN dataset by fuel library. Per each image, i process following steps.

1. Read whole dataset from fuel.
  - a. Fuel combines train/test/extra dataset to one big dataset.
  - b. So I need to redivide train/validation/test set after preprocessing.
2. Calculate smallest boundary that contains all digits.
3. Add padding for preserve original image's width/height ratio.
  - a. If padding area over image's boundary, i expand image's edge pixels.(by numpy's pad function's edge option<sup>17</sup>)
4. Crop image by boundary.
5. Resize cropped image to 32x32 or 52x52.
  - a. I select resizing size by original CIFAR image size(32x32)<sup>18</sup> and Goodfellow's model image size(52x52).
6. Divide datasets to train/validation/test sets.
  - a. Train/validation/test dataset ratio is 90%/5%/5%

In Android App, Just resize cropped image to 32x32 or 52x52.

## Implementation

### Prepare base model

Copy Tensorflow CIFAR-10 tutorial model source code from Tensorflow github. I use most of parameters from tutorial model.

- Convolutional layers filter size : 5x5
- Normalize Options : `tf.nn.lrn(conv, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75)`
- Max pooling stride : 2
- Weight initialization
  - Convolutional layers : `truncated_normal_initializer` with standard deviation of 0.05

---

<sup>17</sup> <https://docs.scipy.org/doc/numpy/reference/generated/numpy.pad.html>

<sup>18</sup> <http://www.cs.toronto.edu/~kriz/cifar.html>

- Fully connected layers : truncated\_normal\_initializer with standard deviation of 0.04

### **Change input/output of base model**

Tensorflow CIFAR-10 tutorial model can accept CIFAR-10 images as input. So, I change that interface can accept preprocessed SVHN images as input. And also, Tensorflow CIFAR-10 tutorial model has just one output. But my new model needs more than 6 outputs(digits length + 5 digits). So, I add digit recognize layers to model.

### **Estimate performance of base model**

First trial is fail. Even can not converge. There are so many trials. I change complexity of neural network and learning rate, batch size. I think too complex model can not show good performance. So, I fix batch size and learning rate fixed at relatively small values. Batch size is 128 and learning rate is 0.1 on Tensorflow CIFAR-10 model. But I think this problem of this model is too high learning rate, I change batch size to 32 and learning rate to 0.01. Finally, I found change optimizer to Adagrad can converge. At this point, base model's minibatch accuracy is 79.7%.

### **Tuning hyperparameters**

Mainly, I test various convolutional layer depths(3 to 8) and shapes. If train and predict result is worse, retry with other depth and shape. Learning rate and batch size is fixed same as base model(0.01, 32).

### **Save Graph**

Save best model's graph definition as file. Using Tensorflow's graph\_util.convert\_variables\_to\_constants and SerializeToString method.

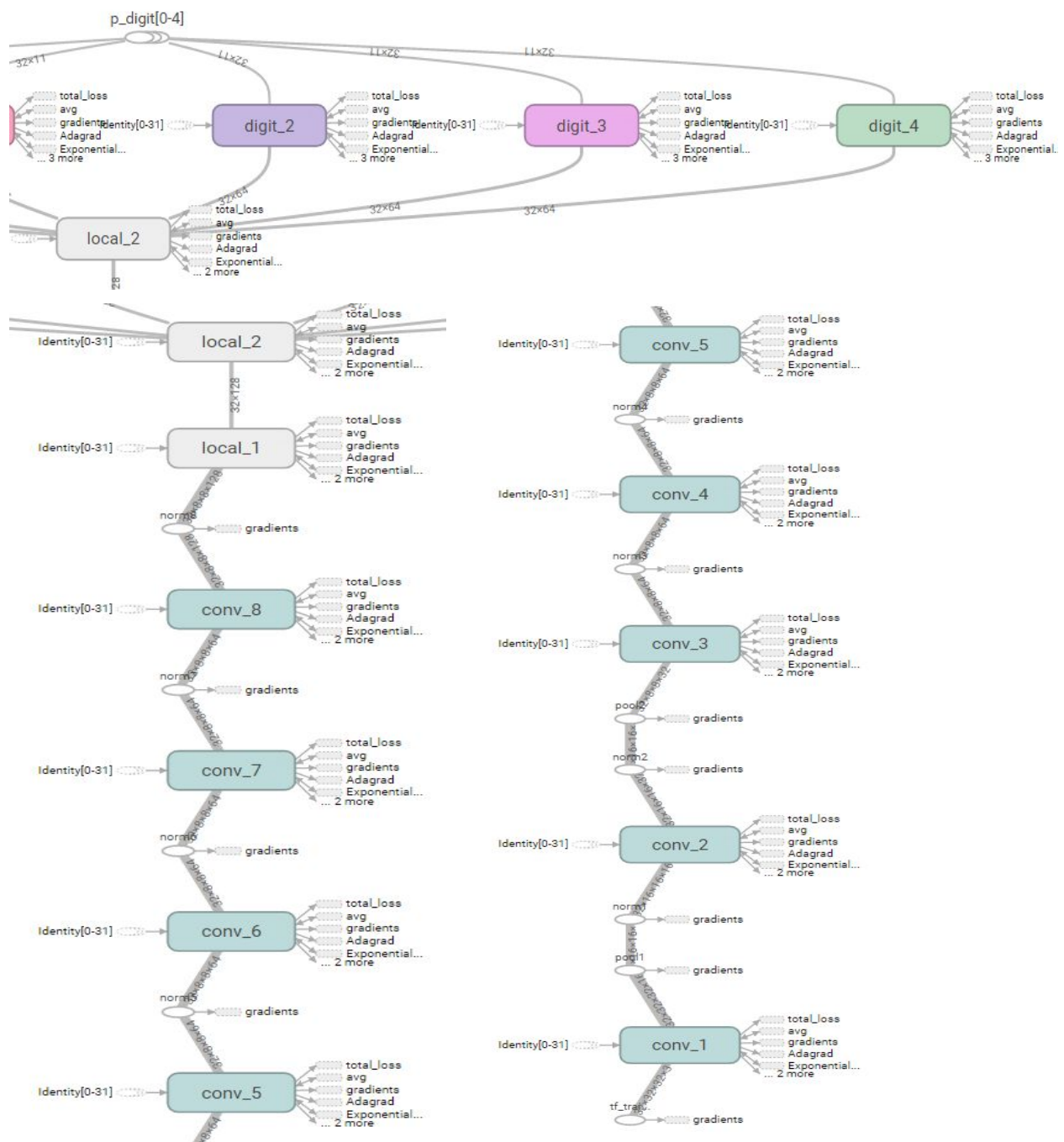
### **Android application**

Android application implements by following steps:

1. Git clone Tensorflow source code from github.
2. Compile Tensorflow android camera demo application.
3. Change input & output logic for multi digit recognition.
4. Copy graph files to assets directory & load that graph.
5. Install to android smartphone & test in real world

**Fig. 7** Final convolutional neural network. Use Adagrad optimizer. The acronyms can be read as follows:

- Conv\_x : x'th convolutional layer
- Local\_x : x'th fully connected layer
- Digit\_x : x'th digit recognize layer



## Refinement

In Benchmark section, I noticed that base model's accuracy is 79.7%. I tune up some parameters following:

1. Convolutional layer count.(3 to 8. Final is 8)

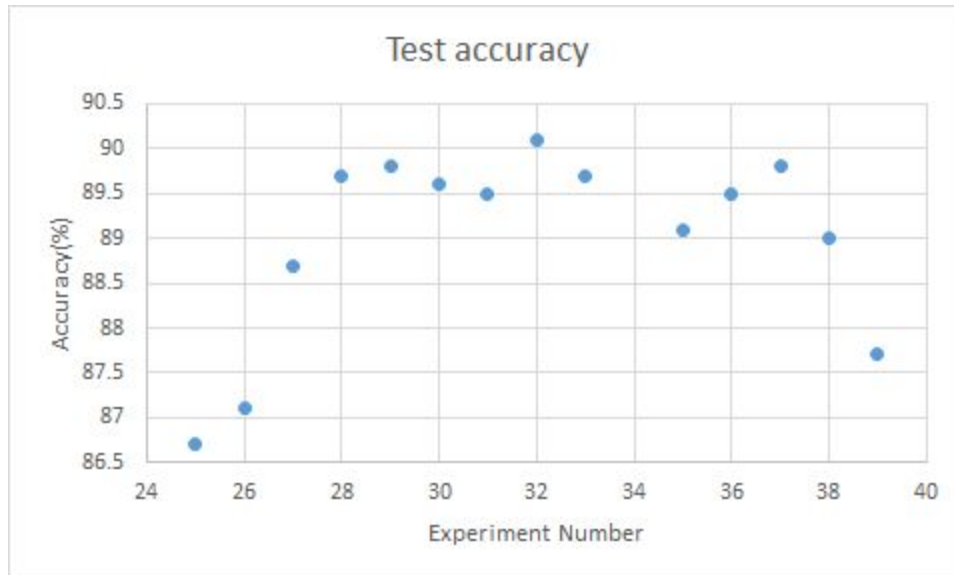
2. Convolutional layer's node count.(8 to 192. Final is [16,32,64,64,64,64,128])
3. Fully-connected layer's node count(32 to 3xxx. Final is [128, 64]).

As a result, best model(Experiment Number 32) has test accuracy of 90.1%.

**Fig. 8** Test accuracy vs. experiment table

Experiment Number	conv_layer_count	conv_layer shape	fc_layer shape	test accuracy
25	3	16,32,64	64,64	86.7
26	4	16,32,64,64	64,64	87.1
27	5	16,32,64,64,64	64,64	88.7
28	6	16,32,64,64,64,64	64,64	89.7
29	7	16,32,64,64,64,64,64	64,64	89.8
30	8	16,32,64,64,64,64,64,64	64,64	89.6
31	8	16,32,64,64,64,64,64,64	128,64	89.5
32	8	16,32,64,64,64,64,64,64	128,64	90.1
33	8	16,32,64,64,64,64,128,128	128,64	89.7
35	8	16,32,64,64,64,64,64,64	64,64	89.1
36	8	32,32,32,64,64,64,64,64	64,64	89.5
37	8	16,32,64,64,64,64,128,128	128,64	89.8
38	8	16,32,64,64,64,128,128,128	128,64	89
39	8	16,32,64,64,128,128,128,128	128,64	87.7

**Fig. 9** Test accuracy vs. experiment



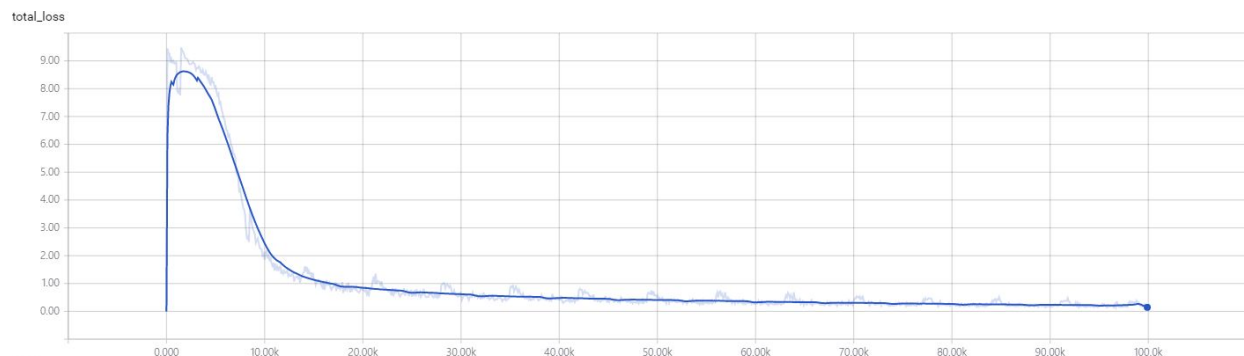
## Results

## Model Evaluation and Validation

Final model's structure & hyperparameters are like this:

- From Tensorflow CIFAR-10 tutorial model
  - Convolutional layers filter size is 5x5(same as Tensorflow CIFAR-10 tutorial model).
  - Each convolutional layer does normalize.
    - `(tf.nn.lrn(conv, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75))`.
  - First and second convolutional layer does max pooling with stride 2.
  - The weights of convolutional layers are initialized by `truncated_normal_initializer` with standard deviation of 0.05.
  - The weights of fully connected layers are initialized by `truncated_normal_initializer` with standard deviation of 0.04.
- Fixed at base model
  - The mini batch size is 32.
  - The learning rate is start from 0.01.
- New values
  - 8 convolutional layer.
    - 8 layer is my experiments maginot line. Too many layer need too many experiments. So I test from 3 layer depth to 8 layer depth, And at best model, 8 layer shows best performance.
  - Node count per each layer is [16,32,64,64,64,64,64,128].
  - First fully-connected layer's node count is 128.
  - Second fully-connected layer's node count is 64.
    - By Fig. 8 and Fig. 9, best model is experiment 32.
  - The training runs for 100,001 iterations(Fig. 10).
    - More training(200,001) can not increase test accuracy.(Fig. 11)

**Fig. 10** total loss graph x axis is iteration.



**Fig. 11** test accuracy vs iterations

experiments number	iterations	loss	mini batch accuracy	validation accuracy	test accuracy
--------------------	------------	------	---------------------	---------------------	---------------

34	200001	0.061928	100	89.6	89.1
35	100001	0.076815	100	89.4	89.1

## Justification

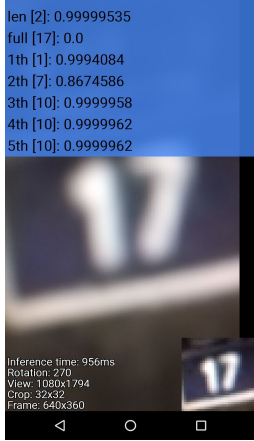
Final model's accuracy is 90.1%. The base model that mentioned at Benchmark section's accuracy is 79.7%. The final model enhance about 10% accuracy. I think it reasonably good.

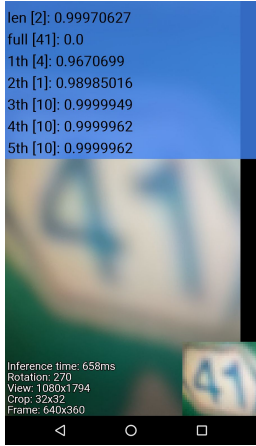
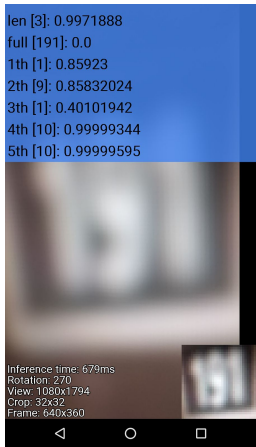
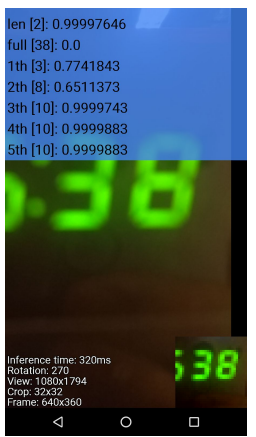
And in android application, it can read digits only digits image boundary is exactly same to image cropping section. After that, it can read digits well. But it is hard to matching image boundary to image cropping section.

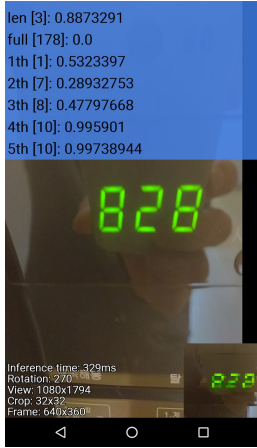
# Conclusion

## Free-Form Visualization

**Fig. 12** Android application digits recognition results. It shows limitation of this app. This app need place camera to digits very closely. Because I train classifier by boundary cropped SVHN model data, It need appropriately cropped at digits boundary image. I'll discuss about this At later improvement section.

 <p>len [2]: 0.99999535 full [17]: 0.0 1th [1]: 0.9994084 2th [7]: 0.8674586 3th [10]: 0.9999958 4th [10]: 0.9999962 5th [10]: 0.9999962</p> <p>Inference time: 956ms Rotation: 270 View: 1080x1794 Crop: 32x32 Frame: 640x360</p>	<p>Digit length : 2 with 0.99999535 confidence Full digits : 17 First digit : 1 with 0.9994084 confidence Second digit : 7 with 0.8674586 confidence Third digit : Unknown with 0.9999958 confidence Fourth digit : Unknown with 0.9999958 confidence Fifth digit : Unknown with 0.9999962 confidence</p>
---	---

	<p>Digit length : 2 with 0.99970627 confidence</p> <p>Full digits : 41</p> <p>First digit : 4 with 0.9670699 confidence</p> <p>Second digit : 1 with 0.98985016 confidence</p> <p>Third digit : Unknown with 0.9999949 confidence</p> <p>Fourth digit : Unknown with 0.9999962 confidence</p> <p>Fifth digit : Unknown with 0.9999962 confidence</p>
	<p>Digit length : 3 with 0.9971888 confidence</p> <p>Full digits : 19</p> <p>First digit : 1 with 0.85923 confidence</p> <p>Second digit : 9 with 0.85832024 confidence</p> <p>Third digit : 1 with 0.40101942 confidence</p> <p>Fourth digit : Unknown with 0.9999344 confidence</p> <p>Fifth digit : Unknown with 0.9999595 confidence</p>
	<p>Digit length : 2 with 0.99997646 confidence</p> <p>Full digits : 38</p> <p>First digit : 3 with 0.7741843 confidence</p> <p>Second digit : 7 with 0.6411373 confidence</p> <p>Third digit : Unknown with 0.9999743 confidence</p> <p>Fourth digit : Unknown with 0.9999883 confidence</p> <p>Fifth digit : Unknown with 0.9999883 confidence</p>

	<p>Digit length : 3 with 0.8873291 confidence</p> <p>Full digits : 179</p> <p>First digit : 1 with 0.5323397 confidence</p> <p>Second digit : 7 with 0.28932753 confidence</p> <p>Third digit : 8 with 0.47797668 confidence</p> <p>Fourth digit : Unknown with 0.995901 confidence</p> <p>Fifth digit : Unknown with 0.99738944 confidence</p>
---	---

## Reflection

This project's process can be summarized as following steps:

1. SVHN data preprocessing.
2. Develop initial model based on Tensorflow CIFAR-10 model and benchmark.
3. Train model and tuning hyperparameters.
4. Develop multi digit recognize Android application based on Tensorflow Android camera application demo.
5. Test Android application in real world.

To me, third step is really hard. Because at first i use Gradient Descent Optimizer. But in that model, total loss is can not converge. After 20~30 times trial, i almost give up. If i don't find AdagradOptimizer is working, i'll be give up this project.

After find that, I test other optimizer also. But in this model, AdagradOptimizer doing best performance.

In Android application, image boundary problem is arise. Because my model trained by boundary cropped images, when using Android application also need feeding cropped image. I think this feature can be added, but need more work.

## Improvement

For convolutional network, there are many improvement points. Add drop-out, tune up normalize hyper parameters, learning rate decay, ... I try some test for adding that feature to my model but result is not really good. So in this time, i can't add these features. But generally, that features can enhance convolutional network performance. So, If i make these model again, i'll try these features also.

As mentioned in above sections, Android application need detecting digits image boundary feature. It can be added by create more complex neural network.

It can be performed on same convolutional network or other convolutional network.