# Capstone Project

**Machine Learning Engineer Nanodegree**

Yongchao Tang

June 19th, 2016

# Definition

## Project Overview

In modern society, various kinds of navigation applications are necessary when people travel and commute, and have played more and more important roles in every aspect of our life. When we look for an address or a store in a map, an incorrect house number may cause to detour and result in delay for some important events. While accurate house numbers in maps are of major importance for users to locate their destinations, adding street numbers to geographic location databases is a challenging task. This is because exact street numbers are hard to collect and sort. Before the advent of navigation applications such as Google Maps, it is easy for us to find a range of house numbers in a zone, but not to spot an exact location for a specific street number. As the demand for the precise navigation of street numbers grows, how to extract street numbers from street view images such as those shot by Google street view cars has become a demanding yet meaningful task.

To be applied in location databases, the accuracy of transcribing house numbers from images should reach the human-level accuracy which is around 98% (Goodfellow et. al 2013). Considering the various sizes, colours, backgrounds and positions, we need to come up with some methods to recognize house numbers.

Machine learning, especially deep learning method, can be a competitive candidate for such a task. Deep learning model does not need explicit programming of the recognition process. The deep learning does not require priori knowledge of the characters in images. The input data to train the deep learning model is the ”Street View House Numbers" (SVHN) Dataset, which is a good large scale dataset collected from house numbers in Google Street View. It holds more than 30K images for the training and 10K for the test. All images contain one to five digits in each house number. Also it has additional 200K images which are different from the train and test set.

## Problem Statement

The target of the project is to design and implement a deep learning model that can recognize and transcribe the whole house number in an image. My expected solution to the problem is to use the combination of deep neural net (DNN) and the convolutionalal neural network (CNN).

A deep neural network (DNN) is composed of multiple hidden layer of units between the input and output layers. Each hidden layer consists of weights and biases, X*W+B = Y, where X is the input matrix coming from previous layer, Y is the output matrix which will be input to the next layer. To accommodate the non-linear classification, each layer also has a non-linear operation to transform the resulting matrix. The network is trained to minimize a loss function which defines the error of the training results compared with the expected targets.

The convolutionalal neural networks (CNN) are inspired biologically from visual system of animals. The visual cells are sensitive to small sub-regions of the visual field. This sub-region is called a receptive field. The receptive fields are tiled to cover the entire visual field, and act as local filters over the input space. We can use a small matrix to represent the local field and do the matrix multiplication between this filter and a patch in an image. The results are the information we extract by the convolutional operations. By moving the local field across the whole image, we actually transform the image into higher-dimension matrix from the original three-dimension (RGB format of images). The CNN can be applied to extract detailed information in an image which should be important for the recognition of sequence of digits.

The whole strategy is to train six classifiers which share the input information but hold independent output layers. The sharing details of images are captured by CNN and input to the six classifiers. One classifier is to recognize the length of digits in an image, while the rest five classifiers are for the digits recognition. The train loss function is the sum of the log probability of each digit and the length of chains, argmax( log( P(s|m) ) ). The prediction of the model can be obtained by calculating the sequence of digits with the largest sum of the log probabilities.

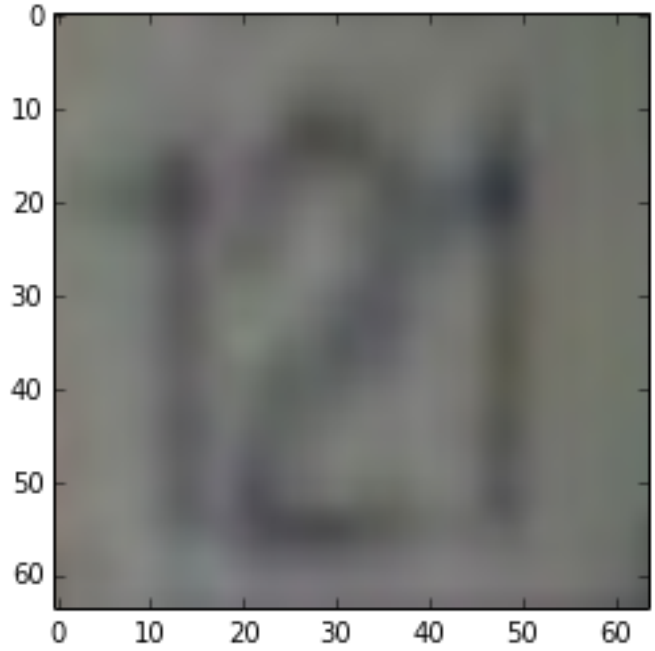
## Metrics

The metric of the model is straightforward and chosen as the accuracy of the predictions of the whole digit chain in each image of test dataset. No partial credit is given to the results because house numbers are helpful to users only when all digits of a house number are transcribed correctly.

# Analysis

## Data Exploration

The SVHN dataset contains more than 30K images of house numbers for the training and more than 10K for the test. There are also an extra images dataset containing more than 200K images for the training if needed. I used the mixture of training dataset and the extra dataset for training. The validation dataset is chosen randomly from the training dataset. The sizes of training, test and validation are 225988, 13068 and 10000, respectively.  One of the examples is shown as follows:



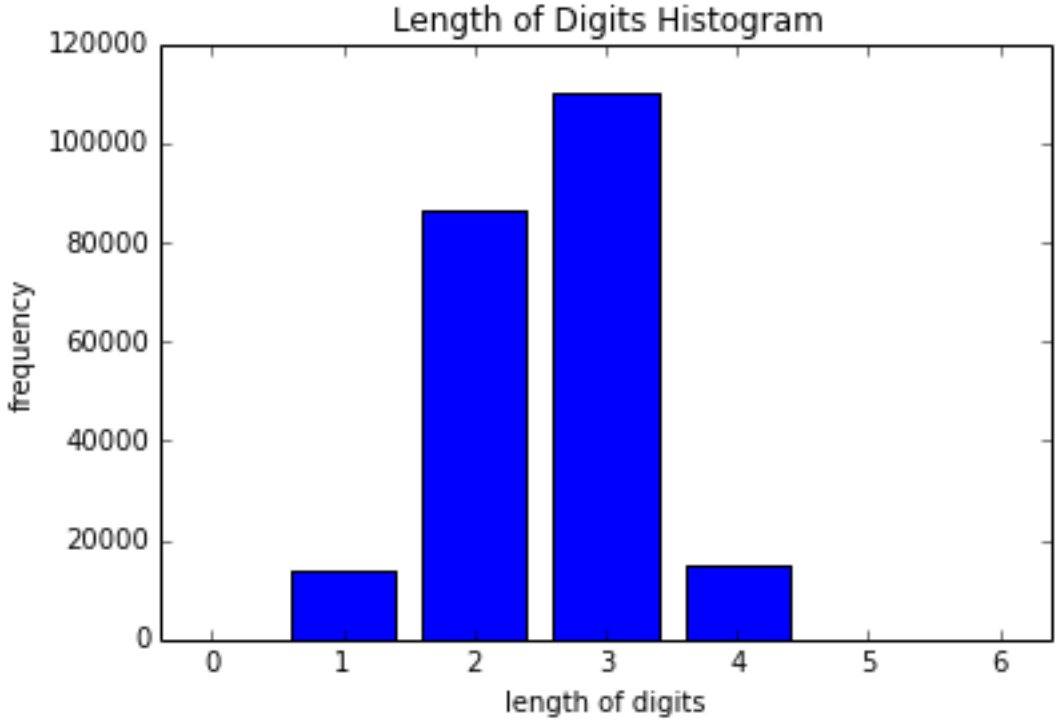

And its pixel values are:

```
[[[130 123 115]
  [127 122 114]
  [125 123 113]
  ...,
  [103 109  99]
  [102 108  99]
  [ 99 104  97]]
```

```
[[130 124 115]
 [128 123 114]
 [125 123 114]
 ...,
 [104 110 100]
 [103 109 100]
 [100 105  98]]

[[130 125 116]
 [127 123 114]
 [125 123 113]
 ...,
 [105 111 101]
 [104 109 101]
 [101 106  98]]

...,
[[119 114 108]
 [122 119 113]
 [125 123 118]
 ...,
 [104 105 104]
 [ 96  99  97]
 [ 91  93  91]]

[[118 114 108]
 [122 118 113]
 [125 122 117]
 ...,
 [ 98  99  99]
 [ 91  93  92]
 [ 86  88  87]]

[[118 114 109]
 [122 118 113]
 [125 122 118]
 ...,
 [ 91  92  92]
```

```
[ 85  87  87]
[ 81  83  83]]]
```

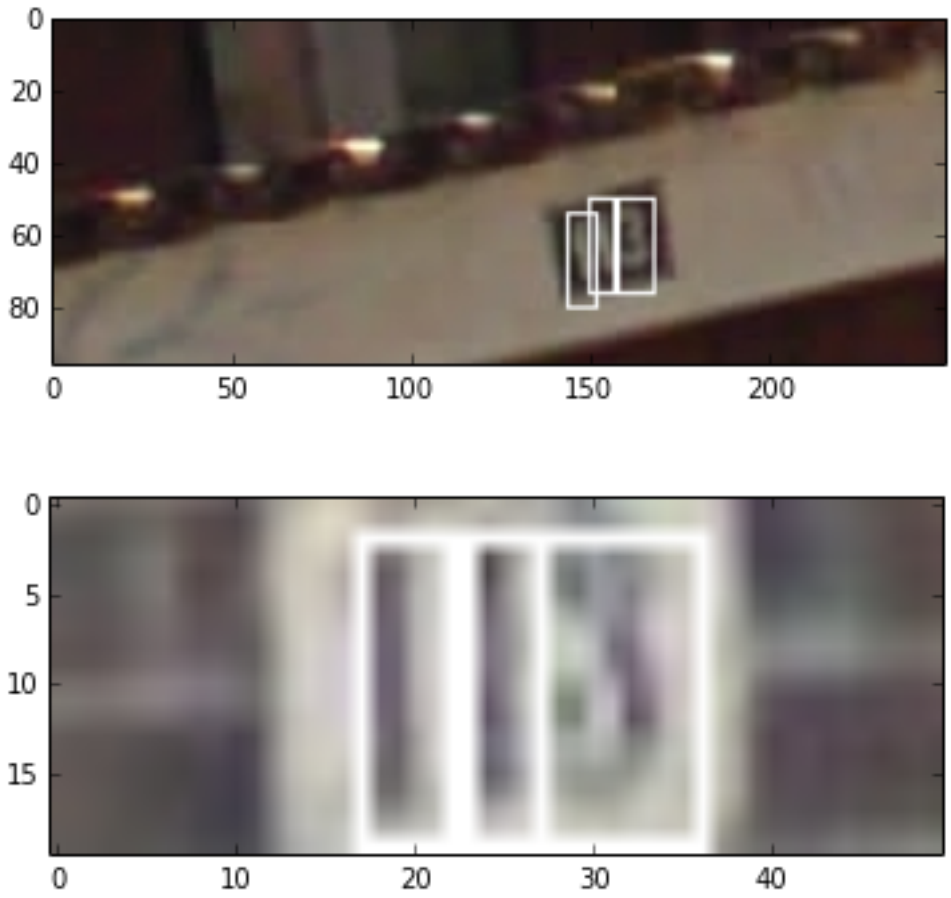
The number of digits in each image ranges from one to six. The distribution of the lengths of digits in the training dataset is:




We can see most of the house numbers in the training dataset are numbers with 3 digits and 2 digits. Thus the training dataset is a highly unbalanced dataset. This is the abnormality I found. Usually we need a balanced dataset to train our model such as SVM or decision trees. However, the CNN deep learning method can handle this difficulty automatically.

## Exploratory Visualization

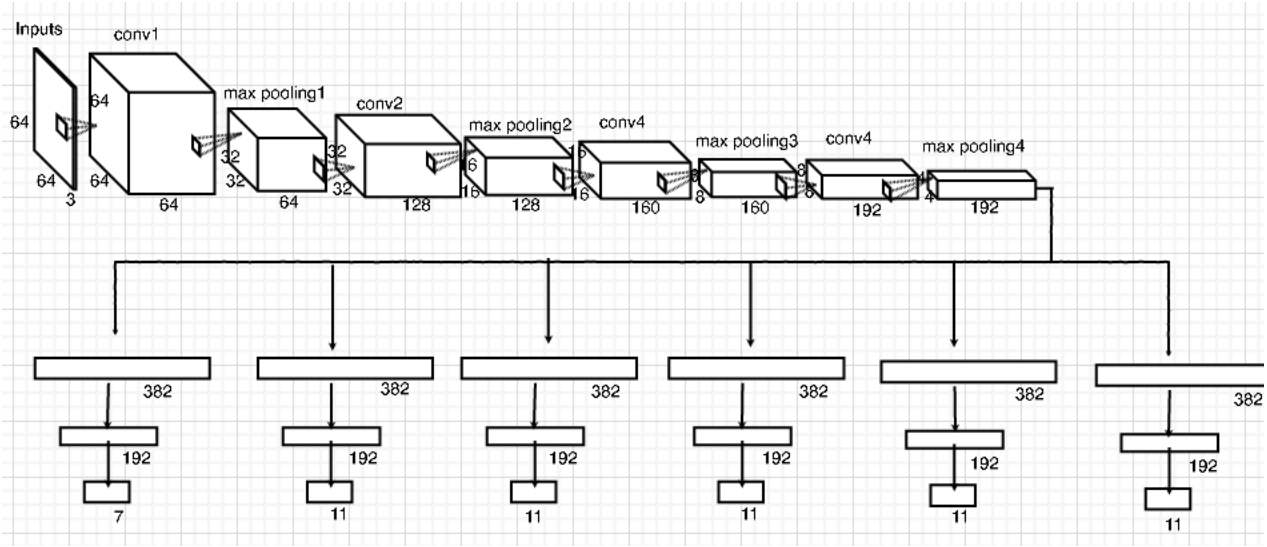
Some original images are shown below:




As it is shown, the images in the training dataset may have thoroughly different sizes and colours. The white boxes for each character is provided by the dataset, which can be used to train an object localizer.

## Algorithms and Technique

My final architecture is composed of four convolutional layers, four pooling layers, and six softmax classifiers.




Four convolutional layers are used to capture the details in an image and output to the six classifiers. A convolutional layer can be seen as a localized connected layer. Only small and localized regions of the inputs are connected to outputs. Each unit in the convolutional layer is connected to a small region of the input neurons or kernel size, for example, a 5 * 5 region, corresponding to 25 input pixels. The region in the input matrix is called the local receptive field for the convolutional layer. And the hidden neutron learns an overall bias as well. We can slide the local receptive field across the entire input matrix. The stride of slides can be determined by users. For example, 1*1 stride means that a local receptive field is moved by one pixel in row or in column at a time. Thus, we will have many local receptive fields. There is a different hidden neutron for each local receptive field in the convolutional layer. The weights for all local receptive fields are same. They are called shared weights and biases. The same weights and biases are used in one convolutional layer to detect exactly the same feature at different locations in the input image. The kernel size I use is 5*5, and the stride is 1*1. Four convolution layers contain [64, 128, 160, 192] units, respectively. To accommodate non-linear feature extraction, non-linear operations are also needed in a convolutional layer. All the non-linear operations are rectified linear units which keep all non-zero values but output zeros when inputs are negative.

Each convolutional layer is followed by a max pooling layer. The max pooling is to choose the maximum of the block they are pooling. The window size for the max pool is 3*3*1, and the stride is 2*2*1.

Since the scale of the weighs in one convolutional layer is important for the convergence of training, some normalization methods are used just before or after pooling layer. Local response normalization is used in my network.

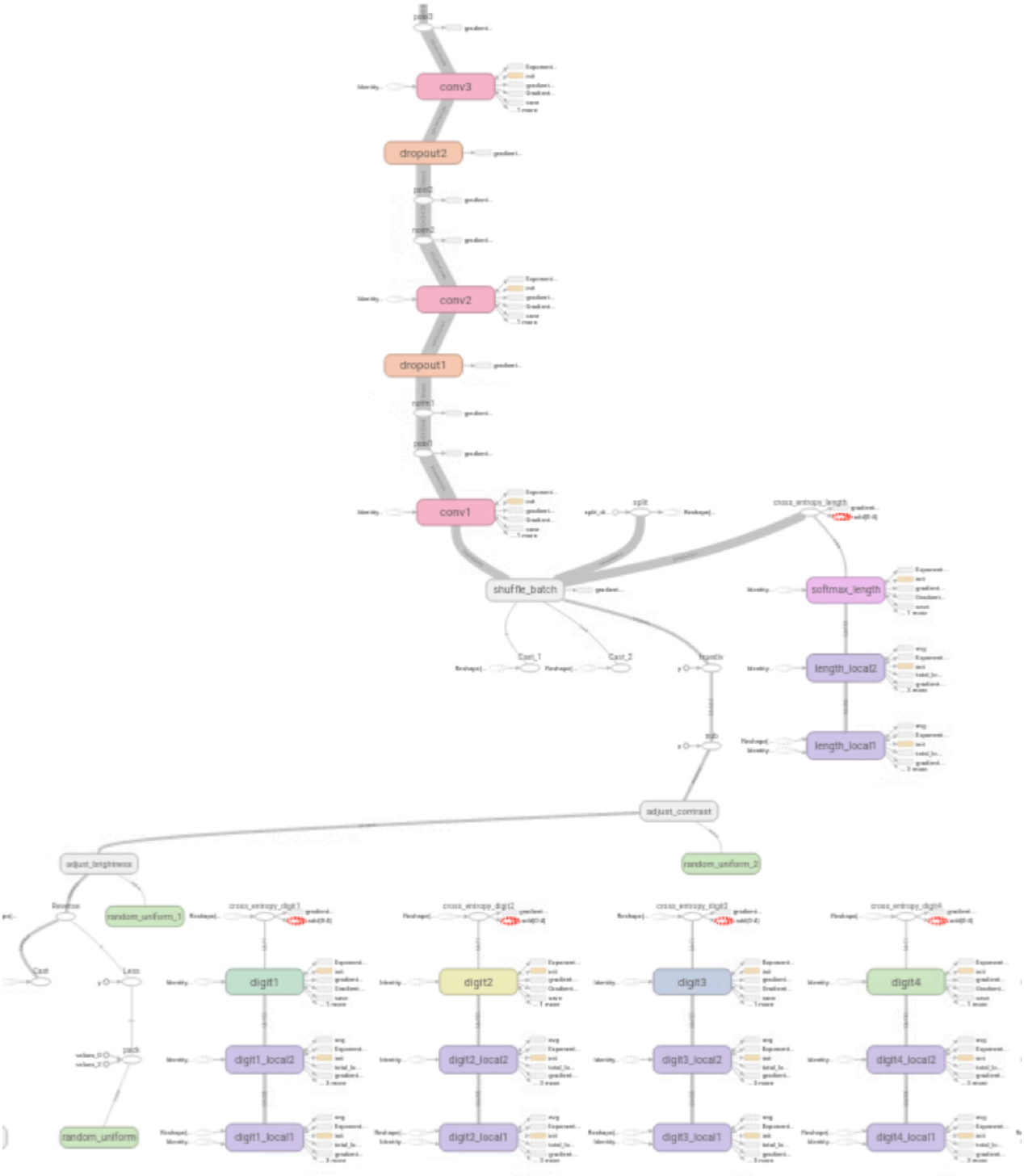
Dropouts are also applied before a convolutional layer except the input layer. The dropout is a kind of regularization which discards connections between input neutrons

and output neutrons at a given chance. Actually it will set some weights to zero at the given possibility.

Every softmax classifier consists of two fully connected layers and one softmax layer. The two fully connected layers have [384, 192] units, respectively. The last sofmax layer is a fully connected layers with 7 or 11 output units. A softmax operation is to map any real numbers to a probability from 0 to 1. It is to calculate the natural exponential of each number and normalize them. No dropout is applied to the classifier layers. The classifier for house number length has 7 classes, which are 0 - 5, and more than 5 digits, respectively. The classifiers for digit recognitions contain 11 classes, 10 classes for 10 digits 0 - 9 and last one is for a blank.

To prevent overfitting, every fully connected layer in each softmax classifier has L2 norm added onto its weights. Adding L2 norm is a kind of regularizations to prevent overfitting. It is the sum of the squares of each value in weights.

I train the model by directly using SVHN. The mini-batch stochastic gradient descent is used. I choose the batch size of 64 by considering the memory size of my computer system.


conv3
dropout2
conv2
dropout1
conv1
shuffle_batch
softmax_length
length_local2
length_local1
adjust_contrast
adjust_brightness
random_uniform_2
random_uniform_1
random_uniform
digit1
digit1_local2
digit1_local1
digit2
digit2_local2
digit2_local1
digit3
digit3_local2
digit3_local1
digit4
digit4_local2
digit4_local1

The graph defined in tensorflow can be visualized with tensorboard. Every operation in the graph will be shown as an elliptical node under the “graphs” tab in the tensorboard. The larger rounded rectangle represents the namespace for one layer. The size of the tensors flowing between two adjacent nodes are also shown. Five classifiers for the recognition of single digits is plotted at the bottom part of the graph, the last one for the length of house number is drawn on top of them.

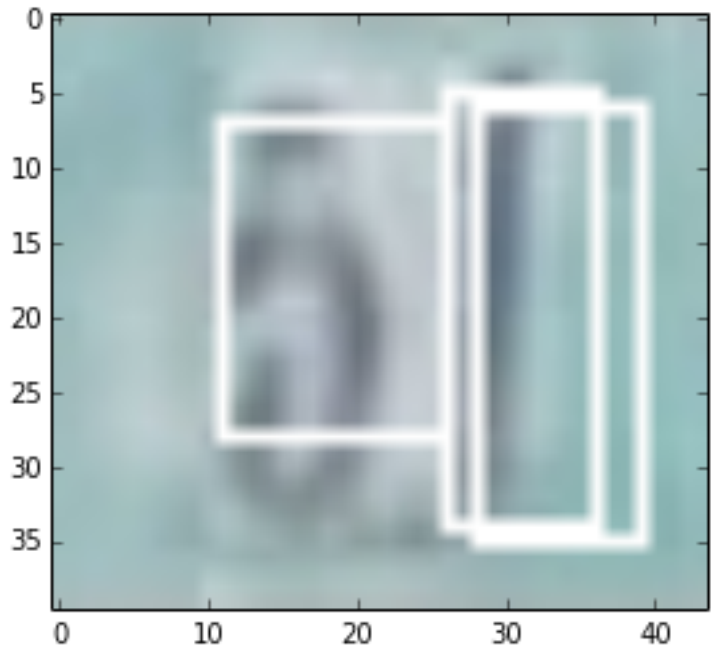
### Benchmark

One existing benchmark is the results published in the paper (Goodfellow et. al 2013). This one should be plausible and achievable made by professional machine learning researchers. Their model achieved 96% for the multi-digit house numbers recognitions.

# Methodology

### Data Preprocessing

An important step before applying the model on the realistic dataset is to preprocess the images of SVHN. There are two reasons to preprocess the dataset: First, there may be some images with wrong labels and wrong digit boxes. For example, the image of 58819.png in the extra folder is labelled with 3 digits, but it actually holds only 2.




/extra/58819.png

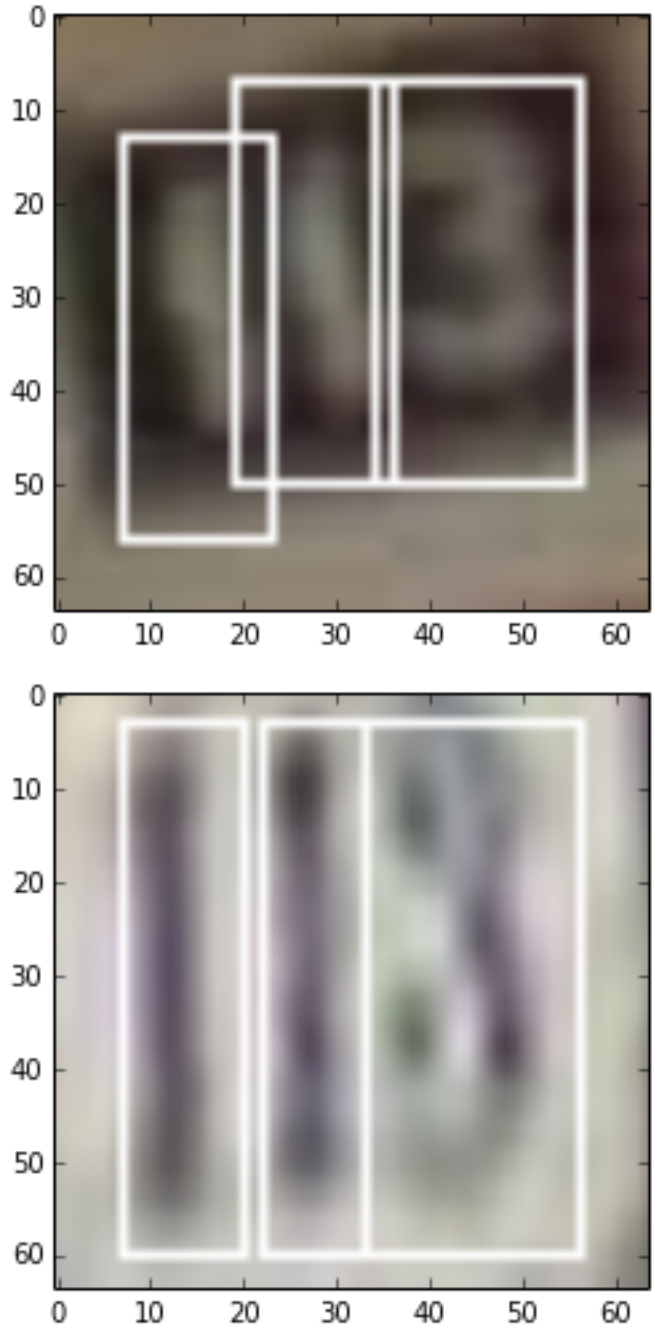
Second, the images in the dataset are not in uniform size. It would be difficult to be imported into tensorflow. To crop them with a same size, I first find the small rectangular bounding box that will contain individual character bounding boxes. I then expand this bounding box by 30% in both the x and the y direction, crop the image to that bounding box and resize the crop to 64 by 64 pixels. This process will induce considerable variability into processed digits sizes since images with same dimension will contain one to five digits.

The following codes are used to resize the images:

```
im = Image.open(image_path)
 (width, height) = im.size
 bbox = digitStruct[i].bbox
 if not isinstance(bbox, np.ndarray):
    bbox = [bbox]
 nBoxes = len(bbox)

 ## box positions
 upper_most = max(bbox[0].top + 1,1)
 lower_most = min(bbox[0].top + bbox[0].height, height)
 left_most = max(bbox[0].left + 1,1)
```

```
        right_most = min( bbox[0].left + bbox[0].width, width)
        for j in range(1, nBoxes):
            upper_most = min( upper_most, max(bbox[j].top + 1,1) )
            lower_most = max( lower_most, min(bbox[j].top + bbox[j].height, height) )
            left_most  = min( left_most, max(bbox[j].left + 1,1) )
            right_most = max( right_most, min(bbox[j].left + bbox[j].width, width) )

        ## enlarge the thumnail area by 30% on x and y axises.
        delta_height_thumnail = (lower_most - upper_most) * 0.3 / 2.
        delta_width_thumnail = (right_most - left_most) * 0.3 / 2.

        ###shrink the thumnail area
        upper_most = max( upper_most - delta_height_thumnail, 1 )
        lower_most = min( lower_most + delta_height_thumnail, height )
        left_most = max( left_most - delta_width_thumnail, 1 )
        right_most = min( right_most + delta_width_thumnail, width )

        thumbnail_size = 64
       thumb = im.transform((thumbnail_size, thumbnail_size), method=Image.EXTENT,
data=(left_most,upper_most,right_most,lower_most), resample=Image.BICUBIC, fill=1)
        thumb.save(save_path, "png")
```

I use the method ‘transform’ in PIL.Image module to resize the images after extending the box containing digits by 30% to include some part of backgrounds.




Two preprocessed images containing the house number “113” are shown above. The coordinates of the bounding box are transformed to the resized images proportionally.

## Implementation

After the images are resized to a uniform size, they are written into a file in the format of serialized string. The serialization can significantly compress the size of the dataset. The length and digits are stored in one-hot form and written to the file.

With the help of the tf.decode_raw method, we can decode the serialization file and import the data into the model. Before input into the model, the images are also distorted by randomly flipped to left or right, added random brightness and contrast in order to increase the size of training dataset. Finally, the values in RGB of every image are subtracted by 128 and then divided by 128. This normalization is straightforward in the live camera android app.

The algorithm generates a batch of images, lengths and digits with batch size 64 from a queue of examples. The data is then input into the model which is defined in the method “inference” in the file “svhn.py”. The convolutional layer has the kernel size 5*5, as proposed in the paper (Goodfellow et. al 2013). The max pooling and normalization operations are widely-used methods in CNN. The max pooling method can make the variances of the elements of a matrix more smooth. The scale of the weights is important for the convergence of the training, thus the normalization of the variables is used to keep the scale in a small range. The dropouts are applied to every convolutional layers. Since the parameters in a deep neural network are on the order of tens of thousands and even more, the model is prone to be overfitted. Discarding some parameters during training steps can make the model more robust against overfitting problem. Each classifier contains two fully connected layers composed of [384, 192] units. These are also usual components  for softmax classifier following the extraction of features from convolutional layers. I do not use 3072 units as suggested in the paper (Goodfellow et. al 2013) because it would be trained too slow on my computer.

The gradient descent optimizer with initial learning rate 0.1 is applied and the learning rate will decay with a factor 0.1 per 100 epochs. The loss value to optimize is the sum of the cross entropies of softmax probabilities of all six classifiers. The L2 regularizations are applied to both two hidden layers in each classifier and are optimized along with the loss value.

## Refinement

At first, I used fixed learning rates 0.1, dropout 0.5, batch size 128, 8 convolutional layers with units [48,64,128,160,192,192,192,192]. The kernel size of each convolutional operation is 5*5. All convolution use zero padding on the input to preserve representation size. Each convolutional operation is followed by a max pooling with 3*3 window size and a local responsive normalization. All fully connected layers in six softmax classifiers contain 3072 units. I used only 33K images to train the model but the model can not be trained because memory is used up.

The training runs out of memory because the batch size is too big. Thus I decrease the batch size to 64. The model began to be trained normally but it seemed too slow to converge.

I think there are too many convolutional layers so I trimmed the 8 convolutional layers to only 2 layers which contain [64,64] units. I added a decay rate 0.1 to the learning rates, and set 100 epochs per decay. The units of two fully connected layers in each classifier are set to [384,192]. This time the model reached an optimum state. The precision on the test dataset is:
step 2000, loss = 18.37 (231.5 examples/sec; 0.276 sec/batch)
accuracy @ 1 = 0.536
step 3000, loss = 10.38 (242.8 examples/sec; 0.264 sec/batch)
accuracy @ 1 = 0.616
step 26000, loss = 3.85 (253.8 examples/sec; 0.252 sec/batch)
accuracy @ 1 = 0.797
step 41000, loss = 3.35 (226.0 examples/sec; 0.283 sec/batch)
accuracy @ 1 = 0.805

Next, I added two more convolutions to the model and changed the number of units [64,128,160,192]. I also alternatively changed the order of pooling op and normalization op in the four convolutional layers. This change could induce more variability in the model and make the model better with fewer parameters. The precision on the test dataset is:
step 8000, loss = 1.58 (230.1 examples/sec; 0.278 sec/batch)
accuracy @ 1 = 0.833
step 15000, loss = 1.52 (224.2 examples/sec; 0.286 sec/batch)
accuracy @ 1 = 0.855

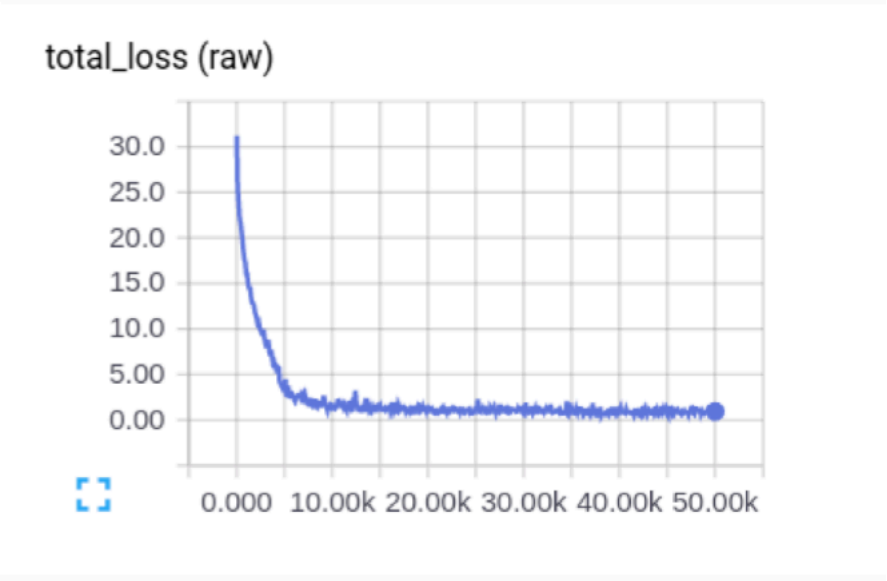
step 56000, loss = 1.07 (228.6 examples/sec; 0.280 sec/batch)
accuracy @ 1 = 0.870
step 131000, loss = 0.26 (231.4 examples/sec; 0.277 sec/batch)
accuracy @ 1 = 0.868

I know dataset size is of same importance to the training of deep learning model as the depth of the model. I combined the extra dataset with the provided training dataset on the website and retrained the model. This time I received a better precision on the test dataset.
step 4000, loss = 3.81 (161.0 examples/sec; 0.398 sec/batch)
accuracy @ 1 = 0.437
step 9000, loss = 1.79 (165.5 examples/sec; 0.387 sec/batch)
accuracy @ 1 = 0.816
step 30000, loss = 0.69 (163.8 examples/sec; 0.391 sec/batch)
accuracy @ 1 = 0.891
step 50000, loss = 1.09 (159.1 examples/sec; 0.402 sec/batch)
accuracy @ 1 = 0.905

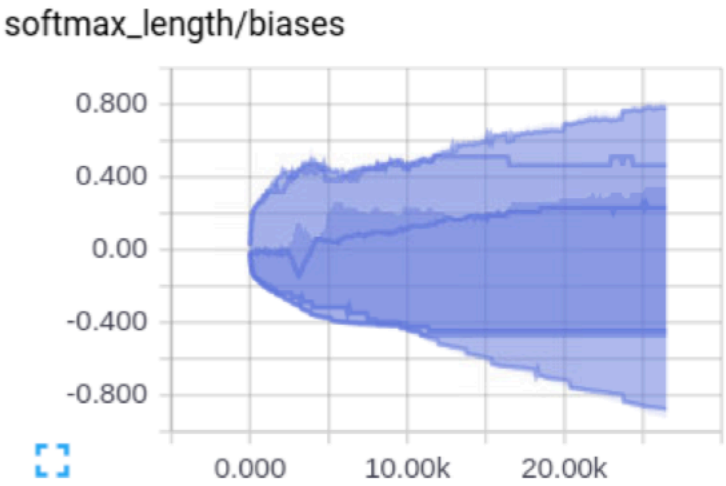
# Results

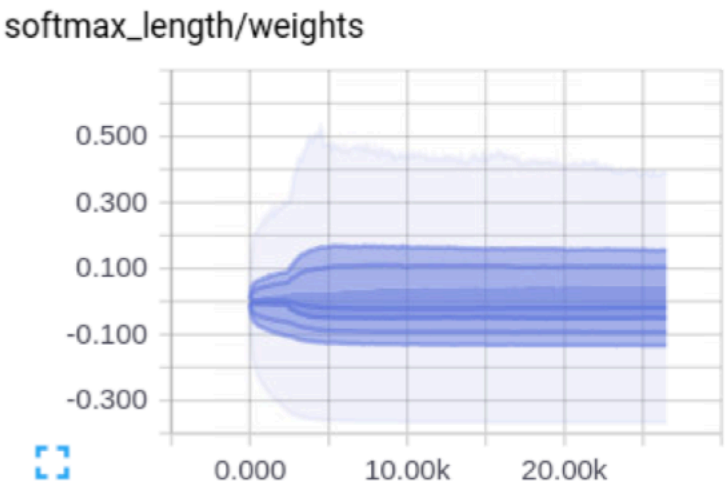
## Model Evaluation and Validation

The training process is early terminated by checking the accuracy of the model on the validation dataset. If the accuracy of the model does not grow on the validation dataset, I train 5K more steps and stop the training. Tuning of hyper-parameters are also based on the validation dataset. My model reaches the accuracy 90.5% at steps 50K on the test dataset of SVHN with 13068 images:
At step 4K, loss = 3.81, accuracy on the test dataset is 43.7%;
At step 9K, loss = 1.79, accuracy on the test dataset is 81.6%;
At step 30K, loss = 0.69, accuracy on the test dataset is 89.1%;
At step 43K, loss = 1.2, accuracy on the test dataset is 90.1%;
At step 50K, loss = 1.09, accuracy on the test dataset is 90.5%;




The total loss levels off after 10K , but the prediction accuracy on the test dataset still grows.




The plots above show the evolution of the weights and the biases of the classifier for the length of digit chains.

## *Android app*

My model is exported to a file with the tensorflow protobuf extension. By importing it into the tensorflow android app example, I can use the model for a live camera stream. The resulting performance of the camera app is not as good as that on SVHN dataset. I guess the down side of the live camera stream is the size of characters. It is not possible for my cellphone camera to capture a steady image with similar digit size as in training dataset.

### *Document how you built the interface to your model.*

The interface to the my model is basically the same as in the original example except the inference process. In the file “tensorflow_jni.cc”, I change the variable “output_names” to std::vector<std::string> output_names ({"output_length", "output_digit1", "output_digit2", "output_digit3", "output_digit4", "output_digit5”}), which includes all names of the output layers of six classifies. After running the model to get the inference probabilities, I call the method “GetTopN” to get all the softmax probabilities exported by the classifier for house number length in descending order, and call the same method to return the digit with highest probability in each digit classifier. By multiplying the probabilities of length and corresponding digits, I can find the most possible house number.

### *Extensions: train a localizer*

To localize where the numbers are in the image, I trained a localizer by using the bounding boxes provided by the SVHN. First, I import the trained model for the house number classification. By making use of the extracted details from convolutionalal network, I built new fully connected layers for the coordinates of each digit. The coordinates of bounding boxes are normalized so that I can train the regression model with the sigmoid_cross_entropy loss function.

## Justification

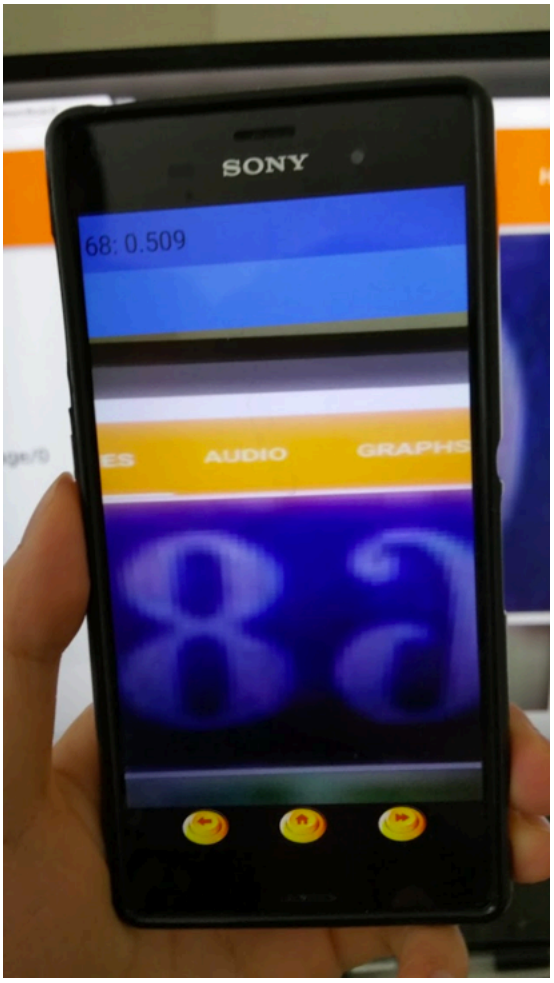
The benchmark I choose is the result published in the paper (Goodfellow et. al 2013). Their best model obtained a sequence transcription accuracy of 96.03%. This is not accurate enough to use for adding street numbers to geographic location databases for placement on maps.

My model reached 90.5% at 50K steps. It further obtained 91.6% at steps 148000. It does not perform better than the benchmark because the depth of my model is far less than the reported model. Deep learning is a heavily computational task. Without powerful computation hardware, it is hardly possible to achieve very high accuracy.

# Conclusion

## Free-Form Visualization

The image below shows how the android app runs after importing my pre-trained model. When the camera of the cellphone captures a picture with a house number, the transcribed house number is shown on the top of the screen, with the corresponding probability on the right.



## Reflection

As a newbie of Tensorflow and deep learning, I met lots of difficulties when trying to write and debug the whole algorithms, such as preprocessing the data, importing the images into model, and saving and restoring the trained model. Also, there are more challenges I had when I try to load the model into the android camera app and train a localizer. I have succeeded overcoming all those challenges and finish the project by running the algorithms smoothly without any bugs. However, the real challenge for me is the tuning of the model.

I am trying to figure out whether it is possible to achieve a higher accuracy by training the limited layers of neural network. Researchers at google can achieve 96% accuracy on similar network structures with sharing weights among independent classifiers. I may try the recurrent network for this task as suggested in the project statement later. But as it is analyzed in the literature, recurrent network is much slower to be trained than convolutional neural network.

There are two end-to-end problems I met during the project. First, Tensorflow c++ api has not as many image preprocessing methods as in the Python api. The tensorflow.image.per_image_whitening method is often called to normalize the input images. However, as no corresponding method existed in c++ api, we can not process images in Android application the same way as on computer. Instead, I choose to subtract each pixel value 128 and divide it by 128. Second, a localizer needs to be trained with fixed boxing coordinates. Therefore, we cannot randomly crop 64*64 images to smaller thumbnails as suggested in the paper (Goodfellow et. al 2013). Random cropping can increase variability of training dataset and prevent model overfitting. The two solutions will surely affect the performance of the final model, but they also made possible the load of the pre-trained model into an Android app and the training of a localizer.

## Improvement

If I had access to more powerful computation resources, I could add more layers and try more different hyper parameters.

## Summary

In this project, I use a convolution neural network as a feature extraction method. The extracted features are shared by six classifiers, one is to predict the length of house number digits, the rest are for the predictions of the single digits. Four convolutional layers, four pooling layers are used in the convolution neural network. The final accuracy of the trained model on the provided test set is 0.905. I input the pre-trained model into an android app and successfully install and run the app on my own

cellphone. I also train a localizer for the bounding boxes of the digits in images.

The whole project has help me understand the process of building a typical convolution neural network. I also grasp some simple techniques for fine-tuning hyper-parameters. With four convolutional layers and total twelve fully connected layers, the model can reach over 90% accuracy. I have witnessed the power of deep learning and become more interested in the development of such a method.