Whole number recognition via RCNN based method

Machine Learning Engineering Nano-degree capstone project

Zhi Zeng

03/01/2017

0. Instruction

All Matlab code are written in Matlab 2016a under Windows 10 environment. I write and run all python code using jupyter notebook in a conda environment under Ubuntu 16.04. In the appendix, all packages in my conda environment are listed.

Be sure to copy all files in the program folder to your working path and modify all path related code in all files before running.

To run the code, first, you have to download the SVHN dataset. Second, use this file to generate the dataset for region proposal network. Third, use this file to generate the dataset for region proposal network. Fourth, use this file to generate the final testing dataset. Fifth, run all the code in this file to obtain the required networks. Finally, run all the code in this file to try the pipline.

To facilitate the users, I have uploaded the final merged model to Google Drive. One can download the file and run the code in this file directly to try the pipeline.

1. Definition

1.1. Project origin

The problem of house number recognition in nature scene is first stated in the article[1] about the SVHN dataset[2]. It is stated that detecting and reading digits from a natural image is an unsolved and hard computer vision task which is central to a variety of real-world applications. Even though highly restricted forms of character recognition are essentially solved problems, recognition digits in the natural scene is more difficult[1]. This is mainly due to non-contrasting backgrounds, low resolution, de-focused and motion-blurred images, and large illumination differences[3].

1.1.1. Related datasets

There are mainly two datasets available that can be used to train a model for digits recognition. The first one is called the MNIST dataset[4,5] which consists of varies handwriting digit images. The other is called the SVHN dataset[2] which consists of approximately 600,000 labeled digits plus 25,000 images containing house numbers. This dataset is captured from Google Street View images and closely related to the problem of number recognition problem in the natural scene. Thus, we use the SVHN dataset in this project.

1.1.2. Related works and project significance

The "Who is the best at X" [6] website provides a crowdsourced list of known results on some of the "major" visual classification, detection, and pose estimation data sets. When clicking on the SVHN category, one may discover that there are approximately 17 related algorithms. All these methods perform well and get an accuracy greater than 95%.

However, most of the above methods are toward the single digit recognition problem. Only a few have announced their performance on the whole number recognition task. In this paper [11], the researcher reported a method to predict both the length of a number and the class of each digit. This method and the like have gained wide applications. However, this method represents the problem as a sequence recognition problem and only works on already extracted number image with a single number. Therefore, how to find the location of each whole number in an image remains a problem. It should be noted that most of these works do not provide pertinent open source code and lack of thorough details to reproduce their score on the SVHN dataset. On the other hand, it is found out that most relevant works do not focus on the number recognition problem only, but on a much broader area of object detection and recognition. Hence, these algorithms are complex and not trimmed for number recognition. Therefore, in this project, some domain knowledge is used to simplify these methods and a concise and robust number recognition program is developed.

1.2. Problem Statement

1.2.1. Problem definition

The problem of number recognition is twofold. First, it should be found out whether there are any digits in the scene and where are the digits. Second, the founded digits should be classified correctly. It should be noted that I do not represent the number recognition problem as a sequence recognition problem because there may be multiple numbers in a real scene.

1.2.2. Strategies

The first task is usually called the object detection and localization. This is the first and most critical step in number recognition since a number can be

correctly recognized only if each digit of the number are detected. The second task is a classification problem. Both classical methods such as the support vector machine and modern methods like convolutional neural networks (CNN) can be used to classify digits.

Recently, region proposal based CNN methods (RCNN) show good performance for object detection, localization, and classification in both accuracy and speed. Related details are well documented in the series of RCNN[7], SPPNET[8], FastRCNN[9], and FasterRCNN[10]. The basic ideas of these techniques are sketched here.

First, the image containing the number is partitioned into many overlapping regions. This partition can also be done in the feature space. Then, a classifier is used to check whether each region contains a digit or not. All digit regions are further merged to several high confident digit regions. Finally, another classifier is used to check which digit is contained in each digit region.

1.2.3. Expected results

It is expected that the accuracy of the classification network is higher than 95%. Since the accuracy of single digit prediction is expected to be 95%, the accuracy of whole number prediction is expected to be 85% provided that all the locations of digits are always predicted correctly and the average length of numbers is 3. However, the datasets for training the region proposal network is not provided by the SVHN dataset and has to be crafted by myself. This indicates that the quality of the region proposal dataset cannot be guaranteed. Thus the accuracy of the region proposal network may be much lower. Similarly, the dataset for training the regression network should also be craft by myself. It is expected that the accuracy of the region proposal network is higher than 90% and that the average error of the regression network is lower than 4 pixels. The overall accuracy for house number recognition is expected to be above 60 percent.

1.3. Metrics

The metric used to evaluate the classification network and the region proposal network is accuracy. Accuracy means the percentage of correct classifications. For example, there are 100 samples to be classified and 75 samples are classified correctly. Then, the accuracy of the classifier is 75%. To evaluate the performance of the regression network, I use the mean average error metric. This metric measures the extent to which the predicted value deviates from the true value in average. For the whole number recognition problem, a number if correctly recognized if and only if each digit of the number is correctly discovered and recognized.

2. Analysis

2.1. Data Exploration

The SVHN dataset contains three datasets. The first one is a training dataset, the second one is a testing dataset, and the last one is called the extra dataset which is much larger and contains relatively simple samples. Each dataset is further divided into two parts. The first part is a dataset contains images of single digits. Figure 1 is a visualization of dataset samples. It can be seen that many samples are not so clear or highly blurred. A few samples are occluded partially.

Figure 1: Sample images of single digits.

The second part consists of images containing full house numbers. All images are in RGB color space. The size of a single digit image is 32x32 pixels, while that of whole number images varies greatly. In addition, not all samples are well allocated. The digits of a number may aline vertically, horizontally, or neither. Figure 2. shows some examples:

I also provide a statistics of the SVHN dataset. In detail, I first count the number of each type of digits and draw the distribution in the form of the histogram. See Figure 3 and 4.

It can be observed that the number of samples in each class is roughly equal. This indicates a roughly balanced dataset.

2.2. Self-crafted datasets

There are three self-crafted datasets. The first one is the dataset used for training the region proposal network. The samples contained in this dataset are

Figure 2: Whole number images.

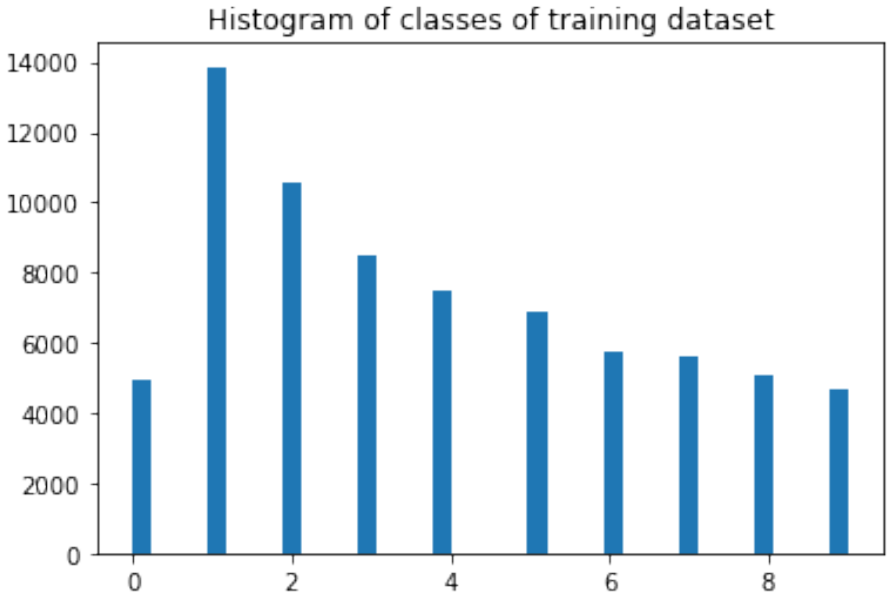

Figure 3: Statistics of the training dataset

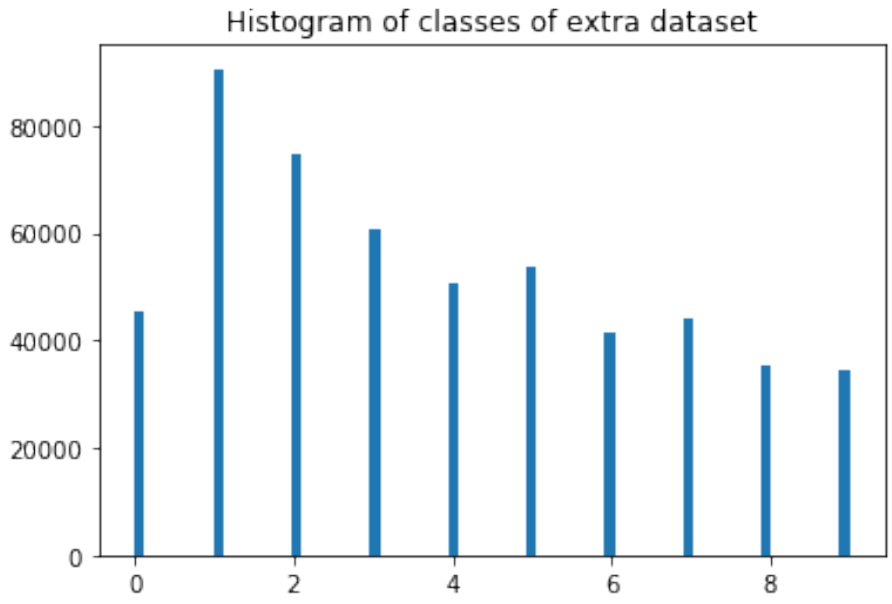

Figure 4: Statistics of the extra dataset

RGB images with a size of 32x32 pixels. The labels are single binary values indicating whether a sample patch contains a digit or not. This dataset is crafted by sampling random regions in images of the full number images dataset. If the sampled region and the ground truth digit regions have an overlapped region over 50%, this region is labeled as true (otherwise as false).

In detail, for each image containing a whole number, I first measure the average height of all digits. Then, I rescaled the image so that the average height of all digits is 32 pixels. Then, I record the bounding boxes for each digit and reduce their size to half around their centers. Candidate regions have the size of 32x32 pixels. If the center of a randomly sampled region is located in one of the reduced bounding boxes, it is labeled as true. This setting guarantee that the overlapping rate is larger than 50%. Figure 5 is a visualization of some samples of this dataset.

One can see from the visualization that this self-crafted dataset does not have sufficient high-quality negative samples which contain rich textures. This may cause false detection of the trained model in the future. Thus, it is expected that the performance of the region proposal network is the bottleneck of the whole algorithm.

The code for generating this dataset is written in Matlab. One can find the corresponding file here.

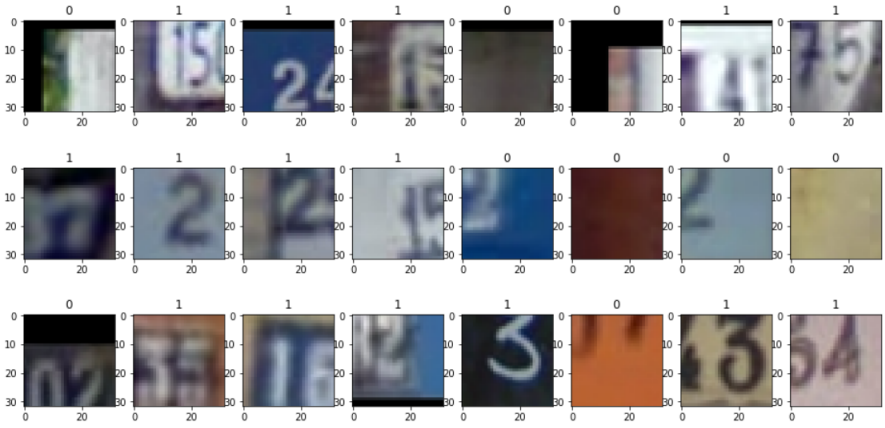

Figure 5: Samples of the region proposal dataset.

The other self-crafted dataset is used to train the regression network. The samples of this dataset are very similar to the positive samples in the region proposal dataset with the labels being the center shifts. Usually, the output of the regression network is the locations of the bounding boxes. However, for the number recognition task, only the center location but not the whole region of a digit matters a lot. Therefore, I only train the regression network for calculating the center shift.

The code for generating this dataset is also written in Matlab and can be found in this file. Figure 6 is a visualization of some samples in this dataset.

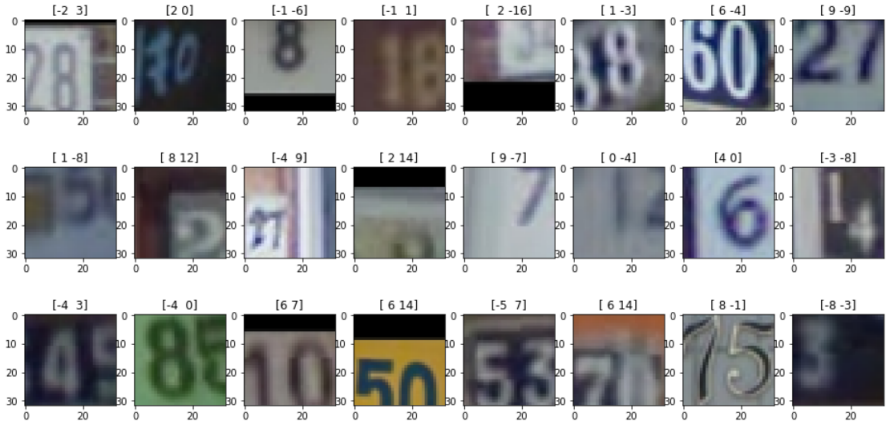

Figure 6: Samples of the regression dataset.

The title of each image is the center shifts along the x and y direction in pix-

els. One can observe that the shift values are not strictly accurate. This is because the original bounding boxes provided by the SVHN dataset are not manually crafted. Hence, this may cause inaccuracy of the regression network. Fortunately, the regression network is used to fine tune the region proposal network. Therefore the inaccuracy of the regression network would not influence the overall performance too much.

The SVHN dataset has also provided us with images containing whole house numbers. However, the sizes of these images are not identical. This contradicts with the requirement that the input of a neural network should be fixed. To solve this problem, I just re-scale each image so that the height of a number is roughly 32 pixels. Then, I pad or crop the images so that all images have the same size of 96x192 pixels. The code for generating this dataset is also written in Matlab and can be found in this file. Figure 7 is a picture illustrating modified samples.

Figure 7: Samples of the modified whole number dataset.

2.3. Algorithms and techniques

The proposed method is mainly based on convolutional neural networks (CNN). CNNs are a category of neural networks that have proven very effective in areas such as image recognition and classification. CNNs have been successful in identifying faces, objects, and traffic signs apart from powering vision in robots and self-driving cars. One could see this webpage for a quick understanding of this technique [12].

In wikipedia, it is stated that the CNN 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. The receptive fields of different neurons partially overlap such that they tile the visual field. The response of an individual neuron to stimuli within its receptive field can be approximated mathematically by a convolution operation [13].

There are essentially four types of layers associated with a CNN: the convolutional layer, the pooling layer, the activation layer, and the fully connected layer.

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 [13].

Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. There are several non-linear functions to implement pooling among which max pooling is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum. The intuition is that the exact location of a feature is less important than its rough location relative to other features. The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters and amount of computation in the network, and hence to also control overfitting. It is common to periodically insert a pooling layer between successive convolutional layers in a CNN architecture. The pooling operation provides a form of translation invariance [13].

Activation layers are used to introduce non-linearity to the network. ReLU is the most commonly used activation layer in CNNs. It is the abbreviation of Rectified Linear Units. This is a layer of neurons that applies the non-saturating activation function f(x) = max(0,x). It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer. Compared to other functions, the usage of ReLU is preferable, because it results in the neural network training several times faster without making a significant difference to generalization accuracy [13].

After several convolutional and max-pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset [13].

2.4. Benchmark result

As is discussed above, the problem is twofold, the first is to find out the location of each digit in an image. The second is to classify each digit. For the first part, there is no benchmark available since no public standard dataset for this task is provided (region proposal for digits only using the SVHN dataset). For the second task, one can find out the state-of-the-art methods with their performances on this website[6]. For the overall performance, there are no benchmark results available.

3. Methodology

3.1. Data Preprocessing

The training data for the classification network, regression network, and the region proposal network are of the same size (32x33 pixels with 3 channels). While the size of the input for the final task is 96x192 pixels with three channels. The input images are normalized to have zero means. The detail is given by the following formula:

```
normalized = image/127.5+1.0
```

I do not perform any contrast enhancement to the training samples. However, one may apply local contrast enhancement (adaptive histogram equalization) to the luminance channel of the input image in Luv (or YUV) color space when testing the performance of the overall algorithm. It is expected that contrast enhancement would benefit the overall performance.

3.2. Implementation

It is discussed earlier, region proposal based CNN methods (RCNN) show good performance for object detection, localization, and classification in both accuracy and speed. I will follow the steps suggested by RCNN[7], SPPNET[8], FastRCNN[9], and FasterRCNN[10] with some modification. In detail, I will construct one CNN for region proposal, regression, and classification respectively. The region proposal CNN is used to detect regions containing a digit. Second, the regression network is used to refine the detection result. Third, the classification CNN is used to label each region containing a digit. Finally, the number is constructed according to the locations and labels of all digits.

It should be noted that the each of these models can be separated into two parts. The first part is called the feature extraction network which is composed of several connected forward CNNs. The feature extraction network can be shared between all three models. This is because the input for the three networks is very similar. The only difference between these models lies in their top layers.

Usually, the top layers are densely connected neural networks. In this project, all top layers of three networks have exactly the same structure except for the final layer. Details regarding these networks are as follows.

3.2.1. Classification network

The feature extraction part consists of five convolution neural networks and four dense connected networks. It should be noted that we replace all densely connected layers with a 1x1 convolution neural network. The functions of the two type of layers are identical except that one does not need to flatten the output of the extraction network when computing the output of the 1x1 convolution layer. This modification is quite useful when computing the classification map of a bigger image since convolution is much faster than the sliding window. Figure 8 shows the architecture of the classification network.

These are the parameters of the network. The code for model construction is contained in section 1.2.1 in the IPython notebook file.

Table I. Summary of the classification network.

Layer (type)	Output Shape	Param #	Connected to
convolution2d_1 (Convolution2D)	(None, 26, 26, 32)	4736	convolution2d_input_1[0][0]
activation_1 (Activation)	(None, 26, 26, 32)	0	convolution2d_1[0][0]
convolution2d_2 (Convolution2D)	(None, 22, 22, 64)	51264	activation_1[0][0]
activation_2 (Activation)	(None, 22, 22, 64)	0	convolution2d_2[0][0]
convolution2d_3 (Convolution2D)	(None, 18, 18, 128)	204928	activation_2[0][0]
activation_3 (Activation)	(None, 18, 18, 128)	0	convolution2d_3[0][0]
convolution2d_4 (Convolution2D)	(None, 14, 14, 256)	819456	activation_3[0][0]
activation_4 (Activation)	(None, 14, 14, 256)	0	convolution2d_4[0][0]
convolution2d_5 (Convolution2D)	(None, 10, 10, 512)	3277312	activation_4[0][0]
dropout_1 (Dropout)	(None, 10, 10, 512)	0	convolution2d_5[0][0]
activation_5 (Activation)	(None, 10, 10, 512)	0	dropout_1[0][0]
convolution2d_6 (Convolution2D)	(None, 1, 1, 512)	26214912	activation_5[0][0]
activation_6 (Activation)	(None, 1, 1, 512)	0	convolution2d_6[0][0]

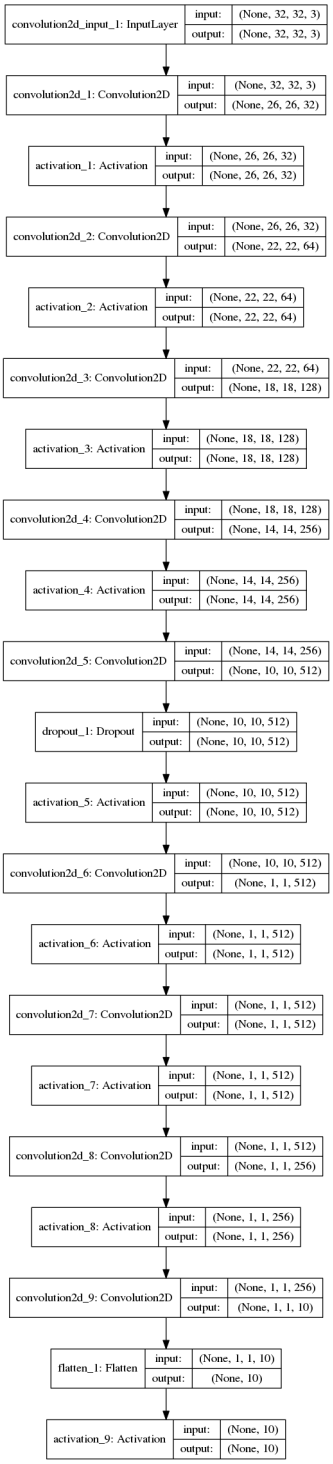

Figure 8: The architecture of the classification network.

```
____________________________________________________________________________________________________
convolution2d_7 (Convolution2D)  (None, 1, 1, 512)    262656    activation_6[0][0]
____________________________________________________________________________________________________
activation_7 (Activation)        (None, 1, 1, 512)    0         convolution2d_7[0][0]
____________________________________________________________________________________________________
convolution2d_8 (Convolution2D)  (None, 1, 1, 256)    131328    activation_7[0][0]
____________________________________________________________________________________________________
activation_8 (Activation)        (None, 1, 1, 256)    0         convolution2d_8[0][0]
____________________________________________________________________________________________________
convolution2d_9 (Convolution2D)  (None, 1, 1, 10)     2570      activation_8[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 10)           0         convolution2d_9[0][0]
____________________________________________________________________________________________________
activation_9 (Activation)        (None, 10)           0         flatten_1[0][0]
====================================================================================================
Total params: 30,969,162
Trainable params: 30,969,162
Non-trainable params: 0
```

The classification network is first trained using the training dataset. To avoid overfitting, I add a dropout layer after the extraction network. The dropout rate is 0.5. In addition, I split the training data set into two parts, a training set and a validation set, the validation split rate is 0.2. I train the network using the Adam method so that I don't have to tune the learning rate manually. The loss is set to be the category cross entropy, and the metric is set to be accuracy. The size of each batch of data for training is 1024. The code for model training is contained in section 1.2.2 in the IPython notebook file. The training result is shown in Figure 9.

Each training epoch costs about one minute. There are ten training epochs in total. Final evaluation accuracy is above 92%. A little overfitting can also be observed.

To make the model more accurate, we also train the model using a much larger dataset, the extra training dataset for five epochs. This data set contains 478017 samples. One can observe very little over-fitting during the training.The final validation accuracy is about 98%. Figure 10 shows the details regarding the training process.

3.2.2. Regression network

Feature extraction part of this network is just identical to that of the classification of work. The only difference between these two networks lies in the last layer of the densely connected the network. The last layer of the classification network has 10 outputs, while that of the regression network has only two outputs. Another difference is that the type of the final activation layer of the classification network is sigmoid, but that of the regression network is linear.

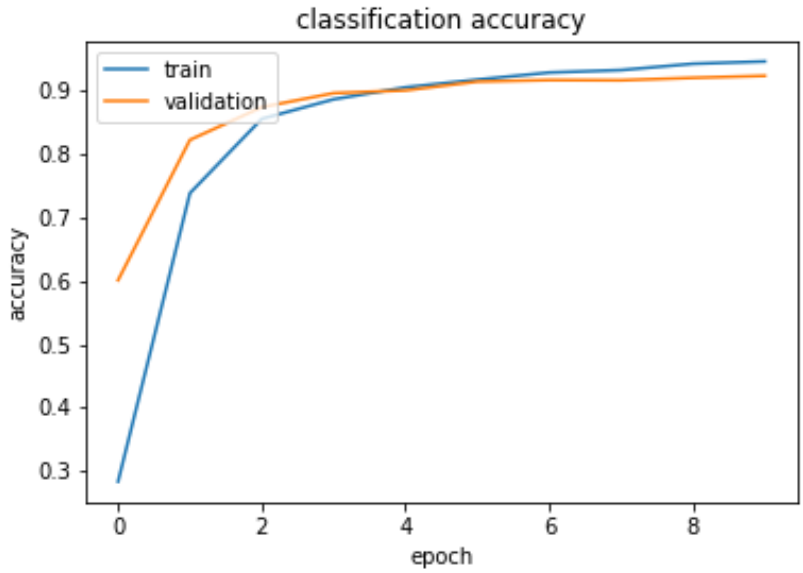

Figure 9: The training history of the classification network.

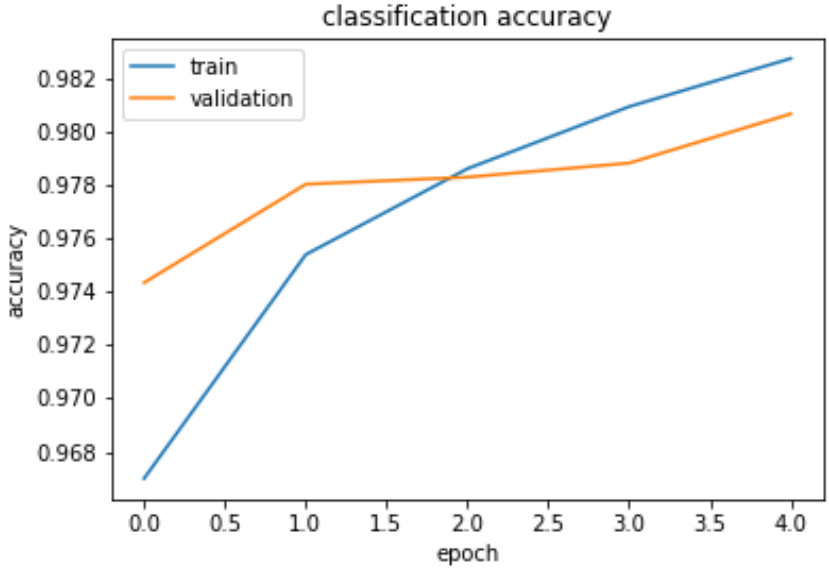

Figure 10: The training history of the classification network.

The code for model construction is contained in section 1.3.1 in the IPython notebook file. Figure 11 shows the details about the settings of this network.

Table II. Summary of the regression network.

Layer (type)	Output Shape	Param #	Connected to
convolution2d_1 (Convolution2D)	(None, 26, 26, 32)	4736	convolution2d_input_2[0][0]
activation_1 (Activation)	(None, 26, 26, 32)	0	convolution2d_1[0][0]
convolution2d_2 (Convolution2D)	(None, 22, 22, 64)	51264	activation_1[0][0]
activation_2 (Activation)	(None, 22, 22, 64)	0	convolution2d_2[0][0]
convolution2d_3 (Convolution2D)	(None, 18, 18, 128)	204928	activation_2[0][0]
activation_3 (Activation)	(None, 18, 18, 128)	0	convolution2d_3[0][0]
convolution2d_4 (Convolution2D)	(None, 14, 14, 256)	819456	activation_3[0][0]
activation_4 (Activation)	(None, 14, 14, 256)	0	convolution2d_4[0][0]
convolution2d_5 (Convolution2D)	(None, 10, 10, 512)	3277312	activation_4[0][0]
dropout_1 (Dropout)	(None, 10, 10, 512)	0	convolution2d_5[0][0]
activation_5 (Activation)	(None, 10, 10, 512)	0	dropout_1[0][0]
convolution2d_6 (Convolution2D)	(None, 1, 1, 512)	26214912	activation_5[0][0]
activation_6 (Activation)	(None, 1, 1, 512)	0	convolution2d_6[0][0]
convolution2d_7 (Convolution2D)	(None, 1, 1, 512)	262656	activation_6[0][0]
activation_7 (Activation)	(None, 1, 1, 512)	0	convolution2d_7[0][0]
convolution2d_8 (Convolution2D)	(None, 1, 1, 256)	131328	activation_7[0][0]
activation_8 (Activation)	(None, 1, 1, 256)	0	convolution2d_8[0][0]
convolution2d_9 (Convolution2D)	(None, 1, 1, 2)	514	activation_8[0][0]
flatten_1 (Flatten)	(None, 2)	0	convolution2d_9[0][0]
activation_9 (Activation)	(None, 2)	0	flatten_1[0][0]

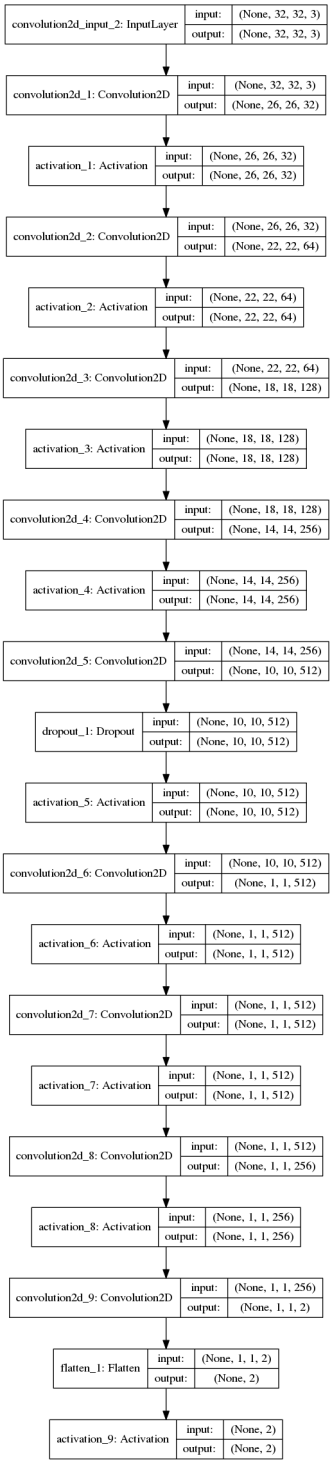

Figure 11: The architecture of the regression network.

```
Total params: 30,967,106
Trainable params: 30,967,106
Non-trainable params: 0
```

We do not have to train the regression network from the beginning since both the classification network and the regression network share the same feature extraction layers. Hence we just copy the weights of the extraction layers of the classification network to the corresponding regression layers before training and fix these layers. This technique is called the transfer learning. The training process of the regression network is very similar to that of the classification network, except that, the losses is set to be the mean absolute error. The code for model training is contained in section 1.3.2 in the IPython notebook file. The training process using the training dataset is given in Figure 12.

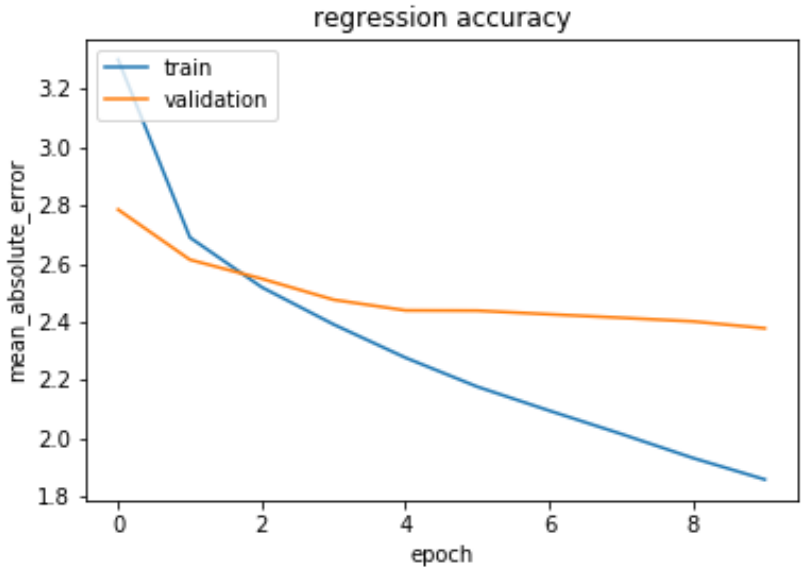

Figure 12: The training history of the regression network.

The final validation loss is 2.3 which indicates that the standard navigation of location prediction is about two pixels. To make the model more accurate, I also train this model on the extra dataset. One could see from Figure 13 that the accuracy has been slightly improved.

3.2.3. Region proposal network

The region proposal network is very similar to the above networks except for the final layer. The size of the output layer is 2 and the activation is sigmoid. The

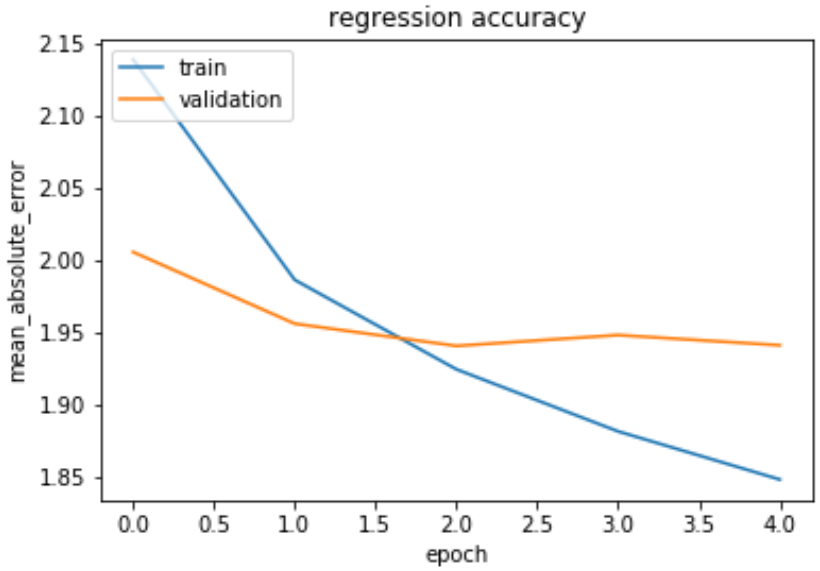

Figure 13: The training history of the regression network.

code for model construction is contained in section 1.4.1 in the IPython notebook file. Figure 14 shows the details about the architecture of the network.

Table III. Summary of the region proposal network.

Layer (type)	Output Shape	Param #	Connected to
convolution2d_10 (Convolution2D)	(None, 26, 26, 32)	4736	convolution2d_input_3[0][0]
activation_10 (Activation)	(None, 26, 26, 32)	0	convolution2d_10[0][0]
convolution2d_11 (Convolution2D)	(None, 22, 22, 64)	51264	activation_10[0][0]
activation_11 (Activation)	(None, 22, 22, 64)	0	convolution2d_11[0][0]
convolution2d_12 (Convolution2D)	(None, 18, 18, 128)	204928	activation_11[0][0]
activation_12 (Activation)	(None, 18, 18, 128)	0	convolution2d_12[0][0]
convolution2d_13 (Convolution2D)	(None, 14, 14, 256)	819456	activation_12[0][0]
activation_13 (Activation)	(None, 14, 14, 256)	0	convolution2d_13[0][0]
convolution2d_14 (Convolution2D)	(None, 10, 10, 512)	3277312	activation_13[0][0]

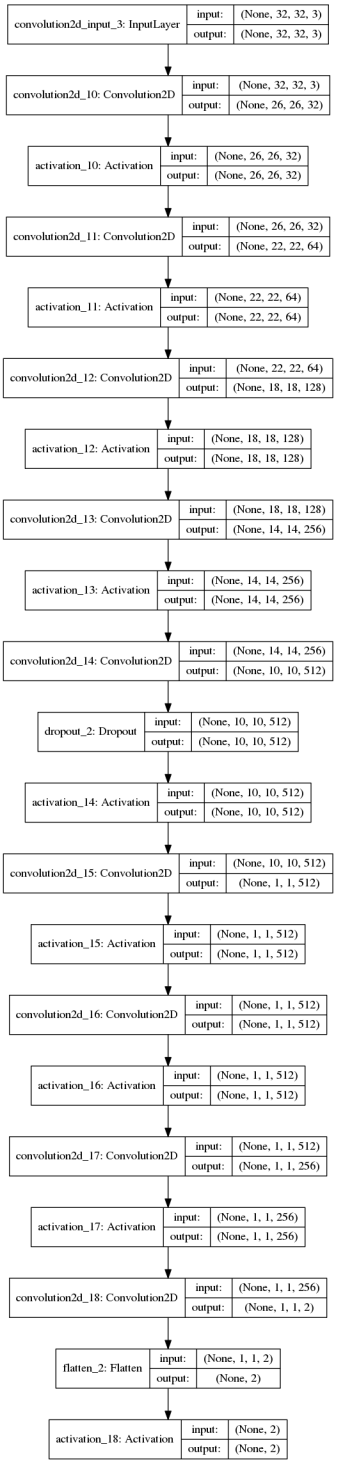

Figure 14: The architecture of the region proposal network.

```
dropout_2 (Dropout)          (None, 10, 10, 512)  0       convolution2d_14[0][0]
activation_14 (Activation)   (None, 10, 10, 512)  0       dropout_2[0][0]
convolution2d_15 (Convolution2D) (None, 1, 1, 512)   26214912  activation_14[0][0]
activation_15 (Activation)   (None, 1, 1, 512)   0       convolution2d_15[0][0]
convolution2d_16 (Convolution2D) (None, 1, 1, 512)   262656    activation_15[0][0]
activation_16 (Activation)   (None, 1, 1, 512)   0       convolution2d_16[0][0]
convolution2d_17 (Convolution2D) (None, 1, 1, 256)   131328    activation_16[0][0]
activation_17 (Activation)   (None, 1, 1, 256)   0       convolution2d_17[0][0]
convolution2d_18 (Convolution2D) (None, 1, 1, 2)     514       activation_17[0][0]
flatten_2 (Flatten)          (None, 2)         0       convolution2d_18[0][0]
activation_18 (Activation)   (None, 2)         0       flatten_2[0][0]
Total params: 30,967,106
Trainable params: 30,967,106
Non-trainable params: 0
```

The code for model training is contained in section 1.4.2 in the IPython notebook file. The training history on the training dataset is shown in Figure 15. One could observe from the result that the final validation accuracy is only slightly above 90% with large overfitting.

To alleviate overfitting, we also train this model on the extra dataset. The following result indicates a little improvement on the validation accuracy. One can see from Figure 16 that the region proposal accuracy is much lower than the classification accuracy. This is why we need the regression network to find tune the result of the region proposal network.

3.2.4 Merge together

Note that the input shape of all these three networks has 96x192 pixels. However, the picture on which we should perform number recognition has a size of 32x32. Hence, to locate each digit, one has to use the sliding window method. However, as mentioned above, the sliding window method is very slow. So, we have replaced all densely connected layers with 1x1 convolution neural networks. Further, we have to modify the input shape of the neural network. The map

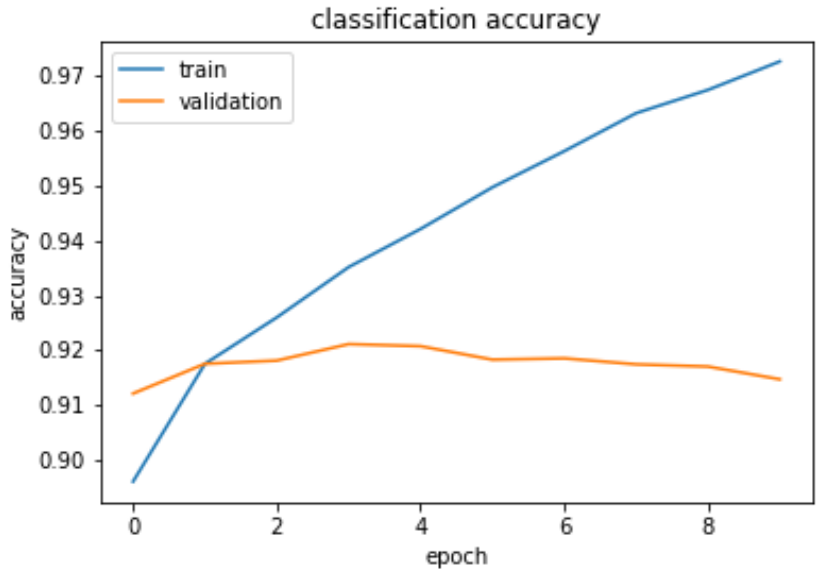

Figure 15: The training history of the region proposal network.

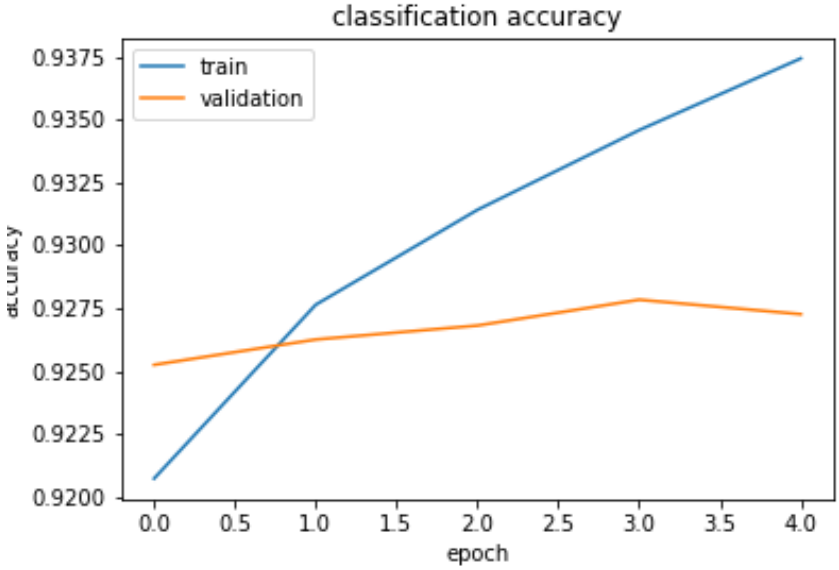

Figure 16: The training history of the region proposal network.

obtained by this modified network is identical to that obtained by the sliding window method. But the proposed method is much faster.

The structure of the model can be further optimized by using a shared feature extractor. How to set up a model with multiple inputs and multiple outputs is well documented in the help section of the Keras web-set. Here, we only provide the graph (Figure 17) and summary of the finally merged multi-output neural network. The code for model construction is contained in section 1.5.1-1.5.5 in the IPython notebook file.

Table IV. Summary of the merged network.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 96, 192, 3)	0	
convolution2d_19 (Convolution2D)	(None, 90, 186, 32)	4736	input_1[0][0]
activation_19 (Activation)	(None, 90, 186, 32)	0	convolution2d_19[0][0]
convolution2d_20 (Convolution2D)	(None, 86, 182, 64)	51264	activation_19[0][0]
activation_20 (Activation)	(None, 86, 182, 64)	0	convolution2d_20[0][0]
convolution2d_21 (Convolution2D)	(None, 82, 178, 128)	204928	activation_20[0][0]
activation_21 (Activation)	(None, 82, 178, 128)	0	convolution2d_21[0][0]
convolution2d_22 (Convolution2D)	(None, 78, 174, 256)	819456	activation_21[0][0]
activation_22 (Activation)	(None, 78, 174, 256)	0	convolution2d_22[0][0]
convolution2d_23 (Convolution2D)	(None, 74, 170, 512)	3277312	activation_22[0][0]
activation_23 (Activation)	(None, 74, 170, 512)	0	convolution2d_23[0][0]
convolution2d_24 (Convolution2D)	(None, 65, 161, 512)	26214912	activation_23[0][0]
convolution2d_28 (Convolution2D)	(None, 65, 161, 512)	26214912	activation_23[0][0]
convolution2d_32 (Convolution2D)	(None, 65, 161, 512)	26214912	activation_23[0][0]
activation_24 (Activation)	(None, 65, 161, 512)	0	convolution2d_24[0][0]
activation_28 (Activation)	(None, 65, 161, 512)	0	convolution2d_28[0][0]
activation_32 (Activation)	(None, 65, 161, 512)	0	convolution2d_32[0][0]

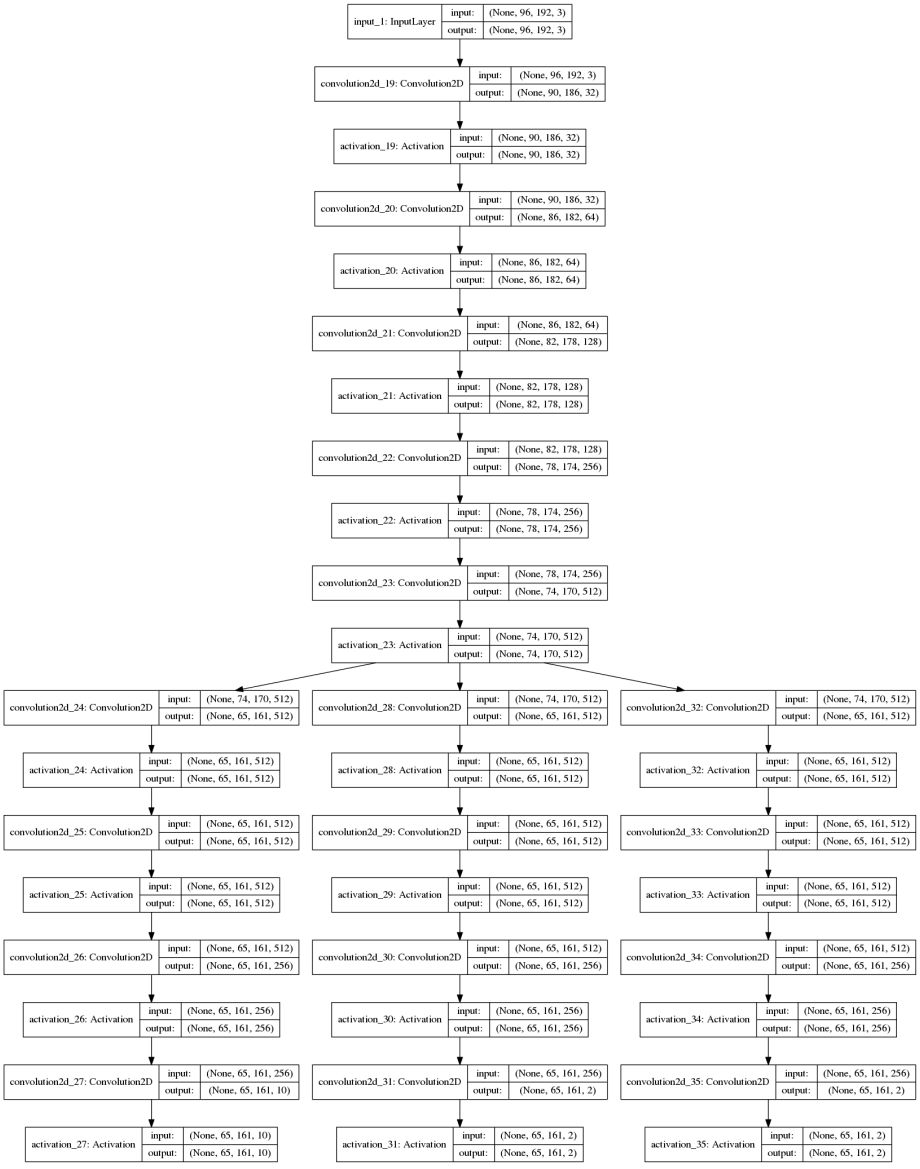

Figure 17: The architecture of the merged network.

```
convolution2d_25 (Convolution2D) (None, 65, 161, 512)  262656    activation_24[0][0]
convolution2d_29 (Convolution2D) (None, 65, 161, 512)  262656    activation_28[0][0]
convolution2d_33 (Convolution2D) (None, 65, 161, 512)  262656    activation_32[0][0]
activation_25 (Activation)     (None, 65, 161, 512)  0       convolution2d_25[0][0]
activation_29 (Activation)     (None, 65, 161, 512)  0       convolution2d_29[0][0]
activation_33 (Activation)     (None, 65, 161, 512)  0       convolution2d_33[0][0]
convolution2d_26 (Convolution2D) (None, 65, 161, 256)  131328    activation_25[0][0]
convolution2d_30 (Convolution2D) (None, 65, 161, 256)  131328    activation_29[0][0]
convolution2d_34 (Convolution2D) (None, 65, 161, 256)  131328    activation_33[0][0]
activation_26 (Activation)     (None, 65, 161, 256)  0       convolution2d_26[0][0]
activation_30 (Activation)     (None, 65, 161, 256)  0       convolution2d_30[0][0]
activation_34 (Activation)     (None, 65, 161, 256)  0       convolution2d_34[0][0]
convolution2d_27 (Convolution2D) (None, 65, 161, 10)  2570     activation_26[0][0]
convolution2d_31 (Convolution2D) (None, 65, 161, 2)   514      activation_30[0][0]
convolution2d_35 (Convolution2D) (None, 65, 161, 2)   514      activation_34[0][0]
activation_27 (Activation)     (None, 65, 161, 10)  0       convolution2d_27[0][0]
activation_31 (Activation)     (None, 65, 161, 2)   0       convolution2d_31[0][0]
activation_35 (Activation)     (None, 65, 161, 2)   0       convolution2d_35[0][0]
==================================================================================================
Total params: 84,187,982
Trainable params: 84,187,982
Non-trainable params: 0
```

3.2.5 The pipeline

The output of the merged network consists of three maps: the classification map, the regression map, and the region proposal map. These maps can be

easily back-project to the raw image plane by padding operation only since the neural network do not contain any pooling layers. The code for these step is contained in section 2.2.1-2.2.2 in the IPython notebook file. Figure 18 is an example image.

Figure 18: Example image.

To visualize the classification map and the region proposal map in a concise way, we label pixels of each class with a unique color. The correspondence between each color and the associated class is shown in Figure 19.

Figure 19: The color bar.

The result of the region proposal network is a binary map. I use it as a mask for the classification map. A typical classification map with region proposal map as its mask is shown in Figure 20. The code for these step is contained in section 2.2.3 in the IPython notebook file.

One can see that the region proposal map may contain noise. Hence, to obtain all digit locations accurately, we need to fine tune the region proposal map. Since digits are seldom overlapped with each other, we basically have two ways to modify the region proposal map. The first one is using the median filter. The second one is using the regression map. To use the first method, one should note that the size of the filter must be chosen carefully, or digits with low contrast or smaller size would be erased. In this project, I use the regression map to fine tune the region proposal map. Details are as follows:

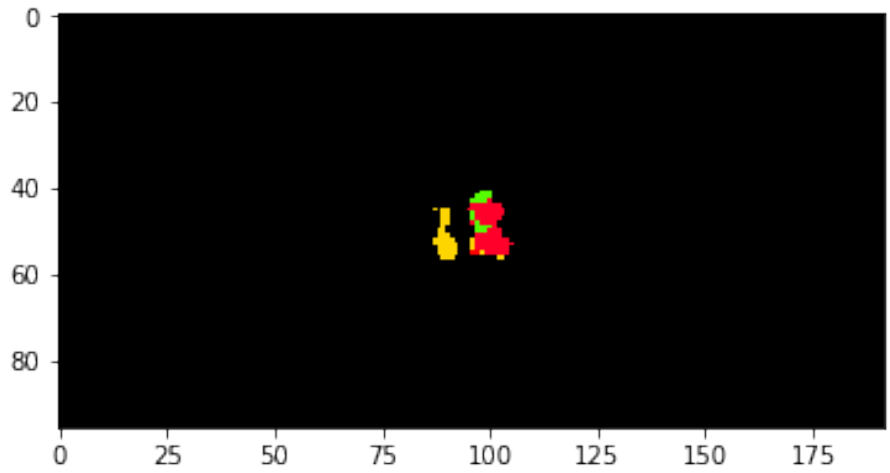

Figure 20: The classification map with the region proposal map as its mask.

First, for each positive pixel on the region proposal map, I move and accumulate the pixel value at a new location according to the shifting value predicted by the regression network. Considering the accuracy of the regression network, before the accumulation, I enlarge the pixel to a patch whose size is SxS pixels, where S stands for the standard deviation of the prediction. After this process, a heatmap is obtained as shown in Figure 21.

Second, I apply a threshold to the heat map, so that an improved mask is obtained. It can be seen in Figure 22 that the modified mask is more accurate that the raw region proposal map. Also, I apply connected component analysis to the improved mask, so that center regions for all digits are clearly separated. The code for these steps is contained in section 2.2.4 in the IPython notebook file.

Third, for each isolated digit region, I count the pixels for each class. Then, the class with most pixels are considered the dominant class for that region. I store both the centers and the classes of all isolated digit regions in a list as follows. The code for these steps is contained in section 2.2.5 in the IPython notebook file.

```
[[ 48.04545455  90.27272727   1.        ]
 [ 49.18181818  99.           7.        ]]
```

Since numbers are usually written from left to right or from top to bottom, I sort and combine the detected digits according to this convention. The code for these steps is contained in section 2.2.6 in the IPython notebook file. Both the detected digits and the numbers are clearly visualized upon the original image as in Figure 23.

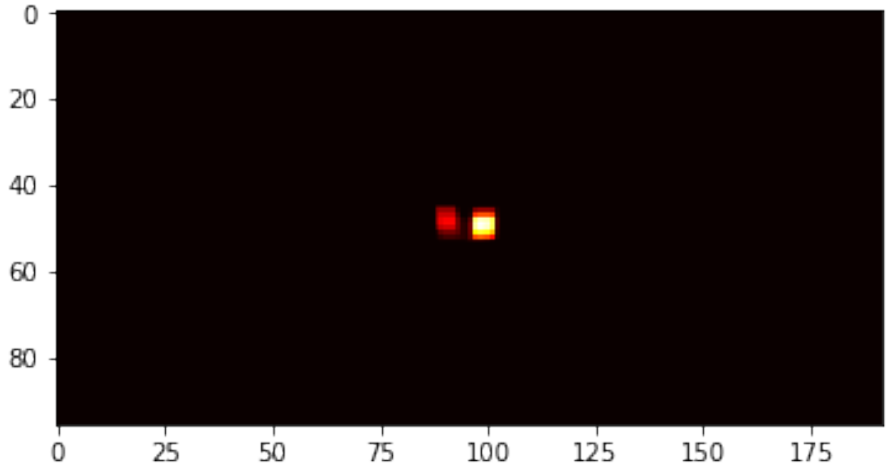

Figure 21: The heatmap.

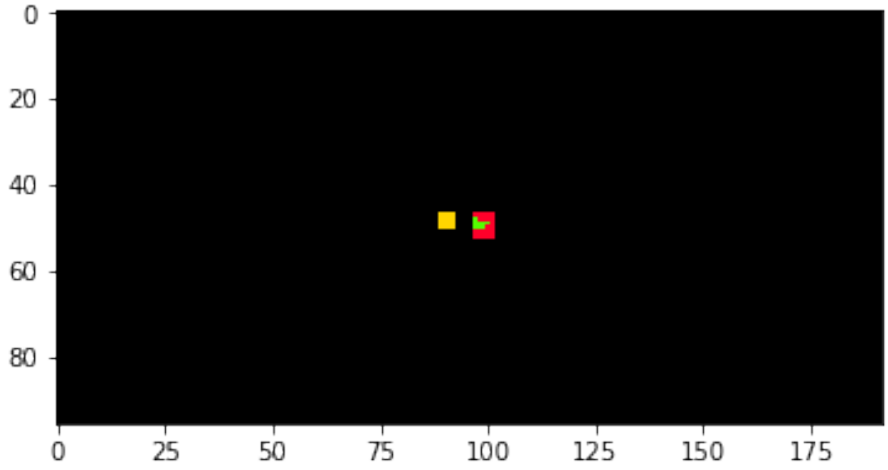

Figure 22: The classification map with the modified region proposal map as its mask.

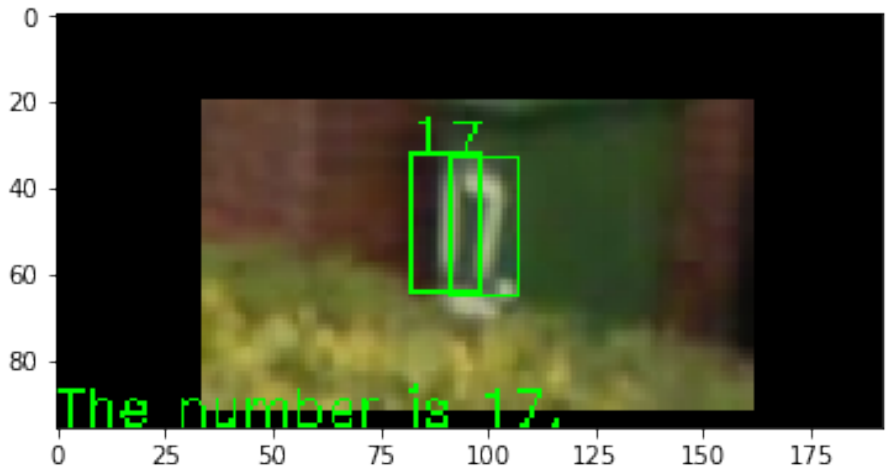

Figure 23: The final result.

3.2.6. Complications in coding

Most of the complications reside in the transfer learning process and the visualization process. After compiling, the Keras network model cannot be modified easily. Modifying the saving model file is not the correct solution either. The only way to perform transfer learning is to set the initial weights of a newly constructed model. The Keras package has already provided us with a visualization tool. However, the tool requires an outdated package, the pydot 1.1.0. Only a few people have the package backups on their Github.

3.3. Refinement and improvement

Actually, there is a long way for me to achieve the goal of this project.

First, the initial program is written in tensor-flow. It is found that modifying an existing model is not so much easy. It cost me too much time to build a program which can be used to modify any part of a model easily. Finally, I rewrite the code using Keras and the final code seems to be more concise.

Second, I had 10 regression networks at the beginning. This means each class of digits has a unique regression network. Such an architecture is suggested by Overfeat[11]. However, it is found in my experiments that multi-regression networks do not improve the final performance significantly.

Third, I did not incorporate a region proposal network at first. But it is found that I can by no means suppress the background noise (the false detection).

Fourth, I did not use the heat map technique in the beginning. Instead, I used

the Gaussian Mixture Model to predict the locations of each digit. However, since it may be difficult to know the maximum length of a number in a real scene, using such a method may hinder the robustness of the system.

Fifth, one may note that the feature extraction network in this project only uses valid padding (no padding). This is because the padded parts do influence the judgment of the neural network. Since I want to discard the sliding window technique and using convolution only, I do not use any padding in the final project.

Sixth, I do not use the region network map directly. Instead, to decrease false detection, I further discard positive pixels whose confidence is lower than 99%. This step can improve the performance a lot.

4. Results

4.1. Model Evaluation and Validation

The neural network of the proposed model has 84,187,982 parameters in total. While the rest of the model has two additional parameters. The first parameter is the threshold for the heatmap. The second parameter is the threshold for the region proposal map. Both parameters are set empirically. In detail, the threshold for the heat map is 20, and that for the region proposal map is 99%. I tried to change the size of each CNN layers and the model got no significant improvement. Adding more layers to the network may lead to in convergence of the model. Therefore, more sophisticated methods such as the residue network or the xception network may be required. However, I have not yet developed a network using these techniques without sliding windows.

I have evaluated the proposed model using the testing dataset provided by the SVHN website. I first evaluated the performance of the classification network, the regression network, and the region proposal network separately. The following result (Figure 24) shows that the accuracy of the classification network is above 95%, while that of the region proposal network is above 92%. The mean absolute error of the regression network is about 2.5 pixels.

Figures 25-27 are some examples of the predictions given by these networks.

I also evaluated the overall performance of the proposed model. The accuracy is 73.4%. Figures 28-30 are some examples of the prediction.

4.2. Compare to the benchmark results

To my knowledge, only the benchmark of the classification network is available. The accuracy of the proposed method is above 95%. However, the most accurate method available can achieve an accuracy above 98%[6]. There is no

1.6.4. classification accuracy

```
In [26]: cls_single.evaluate(np.transpose(np.float32(f_cls_test['X_cls'])/127.5-1.0,(0,3,2,1)),
                             label_binarize(f_cls_test['y_cls'], classes=np.arange(10)))

         26016/26032 [============================>.] - ETA: 0s

Out[26]: [0.2006052443178081, 0.95159803318992009]
```

1.6.5. region proposal accuracy

```
In [27]: rp_single.evaluate(np.transpose(np.float32(f_rp_test['X_rp'])/127.5-1.0,(0,3,2,1)),
                           label_binarize(f_rp_test['y_rp'], classes=np.arange(3))[:,:2])

         52064/52064 [==============================] - 45s

Out[27]: [0.19947120867046791, 0.92117393976644135]
```

1.6.6. regression accuracy

```
In [28]: reg_single.evaluate(np.transpose(np.float32(f_reg_test['X_reg'])/127.5-1.0,(0,3,2,1)), f_reg_test['y_reg'])

         26016/26032 [============================>.] - ETA: 0s

Out[28]: [2.5318032171217739, 2.5318032146306146]
```

Figure 24: Performace evaluations.

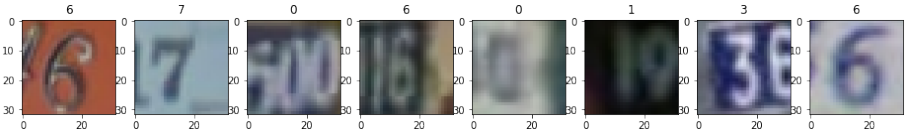

Figure 25: Predictions made by the classification network.

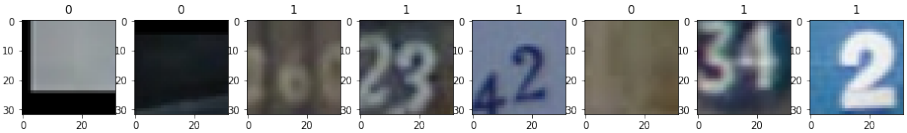

Figure 26: Predictions made by the region proposal network.

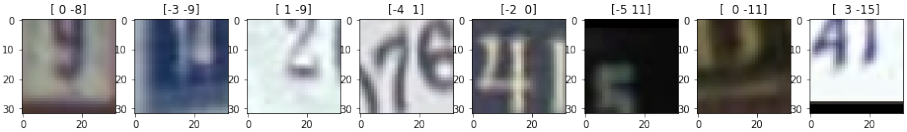

Figure 27: Predictions made by the regression network.

Figure 28: Prediction of '29.png'.

Figure 29: Prediction of '201.png'.

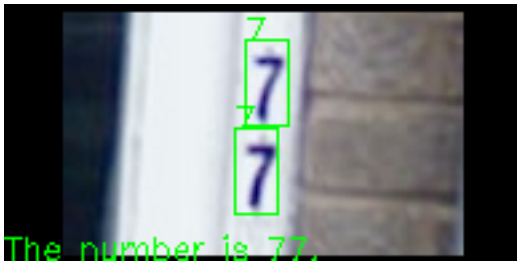

Figure 30: Prediction of '322.png'.

benchmark for the regression benchmark. Fortunately, the performance of the regression network does not dominate the overall performance. Further, the experiments have shown that an average error of fewer than 3 pixels is permissible. The most deficiency lies in the region proposal network. Since one can observe large overfitting in training, it can be concluded that the quality of the training dataset should be improved. Nevertheless, the proposed method has reached a permissible level and has adequately solved the problem defined in the beginning.

5. Conclusion

5.1. Free-form visualization

Here, I want to show more typical samples that are failed to be recognized. They can help me to find out some clues to improving the proposed algorithm. Figures 31-34 are the samples:

Figure 31: Prediction of '8.png'.

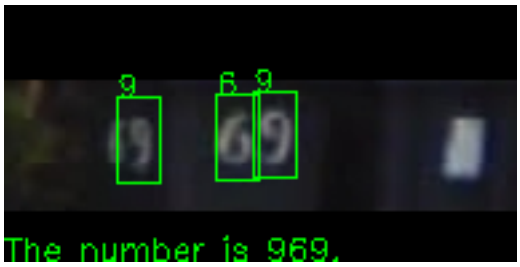

Figure 32: Prediction of '36.png'.

It is found that most errors are due to false positive and negative detection. As discussed before, this is caused by the inaccuracy of the region proposal network, which is caused by the low quality of the manually crafted dataset. Most samples do not contain much confusing texture. Thus, the trained network must not be very discriminative. To alleviate the problem of false negative detection, I should enhance the local contrast.

Figure 33: Prediction of '53.png'.

Figure 34: Prediction of '65.png'.

5.2. Reflection

The proposed method use region proposal based CNN method to recognize house numbers. I use a region proposal neural network to detect the rough locations of all digits and use a regression network to refine the results. Then, I use the result of a classification network together with the connected component analysis to predict the class of each isolated digit region. Finally, I combine all digits together according to their positions.

The most interesting part of this project is that I found the padding technique should be avoided when one want to replace the sliding window method with convolutions only. The second interesting founding is that the quality of the training data is more important than the quality of the model.

The most difficult part of this problem lies in the lack of region proposal dataset. The quality of the manually crafted dataset is hard to control.

5.3. Improvement

First, in this project, I do not consider the cases where a digit is too large or too small or even largely distorted. Hence, the multi-resolution problem is left to be solved in the future. Second, if only one number is permitted in one image, this

problem can be represented as a sequence prediction problem. It is reported that the method based on LSTM and GRU can get much better results.

6. References

[1]. Netzer, Y., & Wang, T. (2011). Reading digits in natural images with unsupervised feature learning. Nips, 1–9. Retrieved from http://research.google.com/pubs/archive/37648.pdf.
[2]. SVHN, http://ufldl.stanford.edu/housenumbers/.
[3]. Sermanet, P., Chintala, S., & LeCun, Y. (2012). Convolutional neural networks applied to house numbers digit classification. Proceedings of International Conference on Pattern Recognition ICPR12, (Icpr), 10–13. http://doi.org/10.0/Linux-x86_64.
[4]. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278–2323. http://doi.org/10.1109/5.726791.
[5]. MINST, http://yann.lecun.com/exdb/mnist/.
[6]. Who is the best at X, http://rodrigob.github.io/are_we_there_yet/build/.
[7]. Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (pp. 580–587). http://doi.org/10.1109/CVPR.2014.81.
[8]. He, K., Zhang, X., Ren, S., & Sun, J. (2015). Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 37(9), 1904–1916. http://doi.org/10.1109/TPAMI.2015.2389824.
[9]. Girshick, R. (2016). Fast R-CNN. In Proceedings of the IEEE International Conference on Computer Vision (Vol. 11-18-NaN-2015, pp. 1440–1448). http://doi.org/10.1109/ICCV.2015.169.
[10]. Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Nips, 1–10. http://doi.org/10.1016/j.nima.2015.05.028.
[11]. Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S., & Shet, V. (2013). Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks. arXiv Preprint arXiv: …, 1–13. Retrieved from http://arxiv.org/abs/1312.6082
[12]. An Intuitive Explanation of Convolutional Neural Networks. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/
[13]. Convolutional neural network. https://en.wikipedia.org/wiki/Convolutional_neural_network

7. Appendix

All Matlab code are written in Matlab 2016a under Windows 10 environment. I write and run all python code using jupyter notebook in a conda environment

under Ubuntu 16.04. This section lists all packages in my conda environment.

```
alabaster                 0.7.10                   py35_0
appdirs                   1.4.0                    py35_0
astroid                   1.4.9                    py35_0
babel                     2.3.4                    py35_0
bleach                    1.5.0                    py35_0
chardet                   2.3.0                    py35_0
cycler                    0.10.0
cycler                    0.10.0                   py35_0
dbus                      1.10.10                       0
decorator                 4.0.11                   py35_0
docutils                  0.13.1                   py35_0
entrypoints               0.2.2                    py35_1
expat                     2.1.0                         0
fontconfig                2.12.1                        3
freetype                  2.5.5                         2
glib                      2.50.2                        1
graphviz                  0.5.2
gst-plugins-base          1.8.0                         0
gstreamer                 1.8.0                         0
h5py                      2.6.0                np112py35_2
hdf5                      1.8.17                        1
html5lib                  0.999                    py35_0
icu                       54.1                          0
imageio                   2.1.2
imagesize                 0.7.1                    py35_0
ipykernel                 4.5.2                    py35_0
ipython                   5.2.2                    py35_0
ipython_genutils          0.1.0                    py35_0
ipywidgets                5.2.2                    py35_1
isort                     4.2.5                    py35_0
jbig                      2.1                           0
jedi                      0.9.0                    py35_1
jinja2                    2.9.4                    py35_0
jpeg                      9b                            0
jsonschema                2.5.1                    py35_0
jupyter                   1.0.0                    py35_3
jupyter_client            4.4.0                    py35_0
jupyter_console           5.1.0                    py35_0
jupyter_core              4.3.0                    py35_0
Keras                     1.2.2
lazy-object-proxy         1.2.2                    py35_0
libffi                    3.2.1                         1
libgcc                    5.2.0                         0
libgfortran               3.0.0                         1
```

```
libiconv                  1.14                     0
libpng                    1.6.27                   0
libsodium                 1.0.10                   0
libtiff                   4.0.6                    3
libxcb                    1.12                     1
libxml2                   2.9.4                    0
markupsafe                0.23                  py35_2
matplotlib                2.0.0
matplotlib                2.0.0              np112py35_0
mistune                   0.7.3                 py35_0
mkl                       2017.0.1                 0
moviepy                   0.2.2.13
nbconvert                 5.1.1                 py35_0
nbformat                  4.2.0                 py35_0
networkx                  1.11                  py35_0
notebook                  4.3.1                 py35_0
numpy                     1.12.0
numpy                     1.12.0                py35_0
numpydoc                  0.6.0                 py35_0
olefile                   0.44                  py35_0
opencv3                   3.1.0                 py35_0    menpo
openssl                   1.0.2k                   0
packaging                 16.8                  <pip>
pandocfilters             1.4.1                 py35_0
path.py                   10.1                  py35_0
pcre                      8.39                     1
pep8                      1.7.0                 py35_0
pexpect                   4.2.1                 py35_0
pickleshare               0.7.4                 py35_0
pillow                    4.0.0                 py35_1
pip                       9.0.1                 py35_1
prompt_toolkit            1.0.9                 py35_0
protobuf                  3.2.0
psutil                    5.2.0                 py35_0
ptyprocess                0.5.1                 py35_0
pydot                     1.2.3
pydot                     1.0.29
pyflakes                  1.5.0                 py35_0
pygments                  2.1.3                 py35_0
pylint                    1.6.4                 py35_1
pyparsing                 2.1.10
pyparsing                 2.1.4                 py35_0
pyqt                      5.6.0                 py35_2
PyQt5                     5.8
python                    3.5.2                    0
python-dateutil           2.6.0                 py35_0
```

```
python-dateutil          2.6.0
pytz                     2016.10                  py35_0
pytz                     2016.10
PyYAML                   3.12
pyzmq                    16.0.2                   py35_0
qt                       5.6.2                         3
qtawesome                0.4.4                    py35_0
qtconsole                4.2.1                    py35_1
qtpy                     1.2.1                    py35_0
readline                 6.2                           2
requests                 2.13.0                   py35_0
rope                     0.9.4                    py35_1
scikit-image             0.12.3                np112py35_1
scikit-learn             0.18.1                np112py35_1
scipy                    0.18.1                np112py35_1
setuptools               27.2.0                   py35_0
simplegeneric            0.8.1                    py35_1
sip                      4.18                     py35_0
sip                      4.19.1                    <pip>
six                      1.10.0                    <pip>
six                      1.10.0                   py35_0
sk-video                 1.1.7                     <pip>
snowballstemmer          1.2.1                    py35_0
sphinx                   1.5.1                    py35_0
spyder                   3.1.3                    py35_0
sqlite                   3.13.0                        0
tensorflow-gpu           1.0.0
terminado                0.6                      py35_0
testpath                 0.3                      py35_0
Theano                   0.8.2
tk                       8.5.18                        0
tornado                  4.4.2                    py35_0
tqdm                     4.11.2
traitlets                4.3.1                    py35_0
wcwidth                  0.1.7                    py35_0
wheel                    0.29.0                   py35_0
widgetsnbextension       1.2.6                    py35_0
wrapt                    1.10.8                   py35_0
xz                       5.2.2                         1
zeromq                   4.1.5                         0
zlib                     1.2.8                         3
```

8. Acknowledge

I thank X. Z. Wang for the discussions and helps about the initial framework of this project. I also thank the reviewer for their efforts.