Introduction

To create an accurate map, finding a way to recognize houses number in photographs is essential. Although the problem definition is pretty straightforward and the goal is simply just recognizing numbers in picture but this is a difficult task because there are many different visual appearance of numbers in terms of digits' orientation, color, shades, brightness and precision of picture. There have been many approaches to capture digits out of a photograph. A traditional approach was a three step process, localization, segmentation and recognition. In this algorithm the goal is to find each digit's location, build a segmented pattern of all digits and finally recognizing theme. In other words, in this approach, the goal is to find each digit individually and then based on their location find the order of the digits to make the whole number. But this paper's approach is inspired by the algorithm described in a paper written by a Google team, "*Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks"* [1]. In this approach in contrast to the traditional one we do not follow a step by step process but we want to combine all the process together. To reach this goal we will utilize a Convolutional Neural Network to get a probability distribution of each digit's chance of occurrences in the photograph and also the probability of the length of the number. We have used two convolutional layer with 8 and 16 filters respectively. In this algorithm we will feed the house numbers images to the network and will get 7 outputs assuming that the number length is not more than 5 digits. Although we will get the probability of the length more than 5, we are not able to recognize digits in the 6th place or more. The big picture of these approach is shown in Figure 1

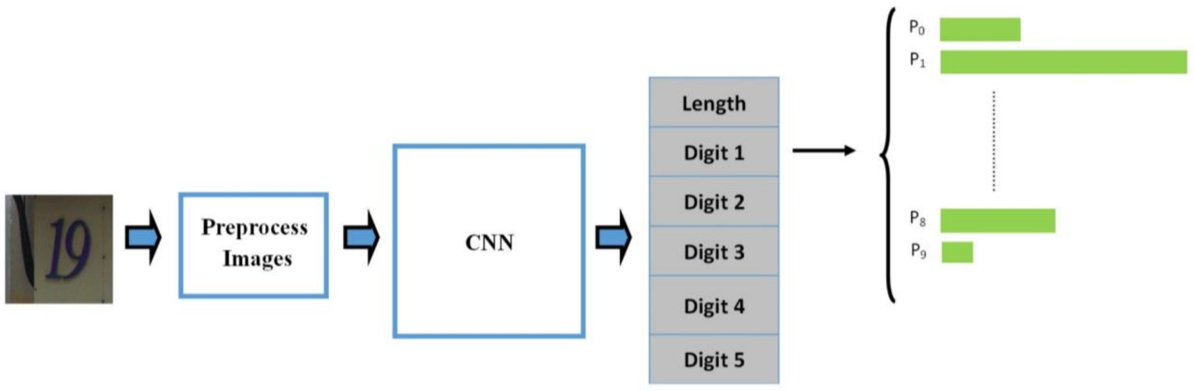

Figure 1. flowchart of the model

Last column represents the probability distribution of digit one which in this example label 1 is the right answer as the first digit in number 19

As shown in the figure above we can see that this algorithm includes 3 general steps. In the first step we will process all the training images (explained in part ‘Dataset’), Feed the processed inputs to a convolutional neural network and finally get a probability distribution for each digit and the length of the number (explained in part ‘Architecture).

Dataset

We have used SVHN (street view house numbers) dataset obtained by Google Street View images [2]. In our experiment we’ve used 33,402 pictures for training data and 13,068 for test data. These data set comes with a separate file which includes each image’s name, label and size. You can see a sample of pictures in Figure 2

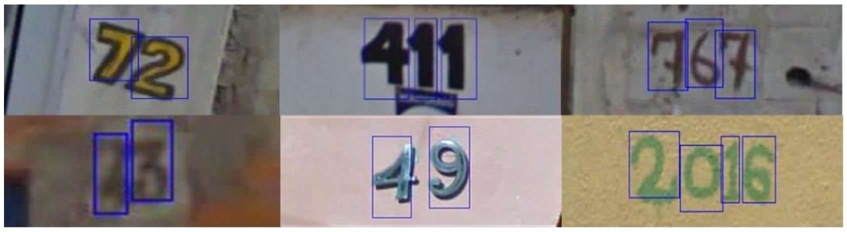

Figure 2. Sample set of pictures form training dataset

Preprocessing data

The high dimensionality of input features which in our case is the number of pixels is very essential problem to overcome, otherwise, it will cost more time and more memory. Look at the images above that are samples of our dataset, you can see that almost 2/3 of the image space is blank and is not part of the number we want to detect, so we cropped the numbers out of the picture and by doing this we have reduced size of the pictures significantly. And because color doesn’t really matter to detect a digit we changed the RGB pictures to Grayscale format. Finally, to reduce the dimensionality of input features the resolution of pictures is changed to 28x28. In Figure 3 you can find the flowchart of preprocessing of pictures with an example. So we can summarize this procedure into 3 phases:

1) Cropping images using the “*bbox*” struct provided in the data set which includes: size, name and each digit bounding box.
2) Changing image to Grayscale
3) Changing resolution to 28x28

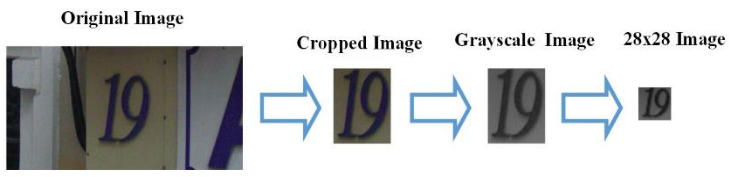

Figure 3. Flowchart of preprocessing images

Now that we have all pictures transformed to the desired format, we will save each picture in a flattened vector of size 28*28 = 784 and also saving the label in on-hot encoding vectors which its length is equal to number of classes. Each *Digit* label has the length of 10 (0-9) and each *Length* label has the length of 7 (0-5 and 'greater than 5'). In other words, our classes are divided into two categories:

1) Digit Label

0	1	2	3	4	5	6	7	8	9

2) Length of Number

0	1	2	3	4	5	>5

Since in this problem we need to find more than one label mapped to an input, then we saved each label in a separate n-dimensional array and consequently we'll end up with five arrays with shape of (10, number of dataset) and one array of shape (7, number of dataset) like Figure 4. We saved these transformed data into 2 pickle files: data.pickle and test_data.pickle for train and test datasets respectively.

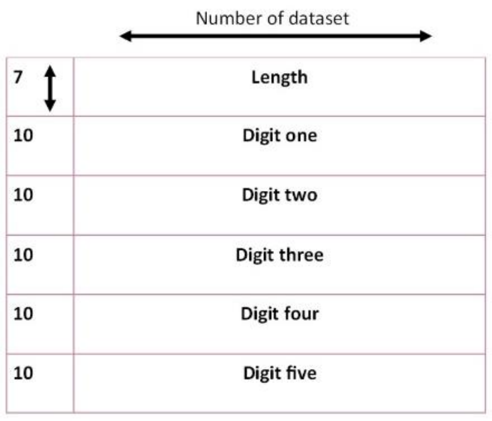

Figure 4. final shape of data

Data Exploration

Figure 5 shows that labels distribution of our training datasets. We can see that except labels 1 and 2 for the rest of the data set distribution is mostly uniform. But we should consider this fact that each input data is a combination of labels namely digits. And in the real world data, contribution of digits 1 and 2 is significantly higher than other digits. Therefore, we can claim that our data set represents a precise sample of the real world.

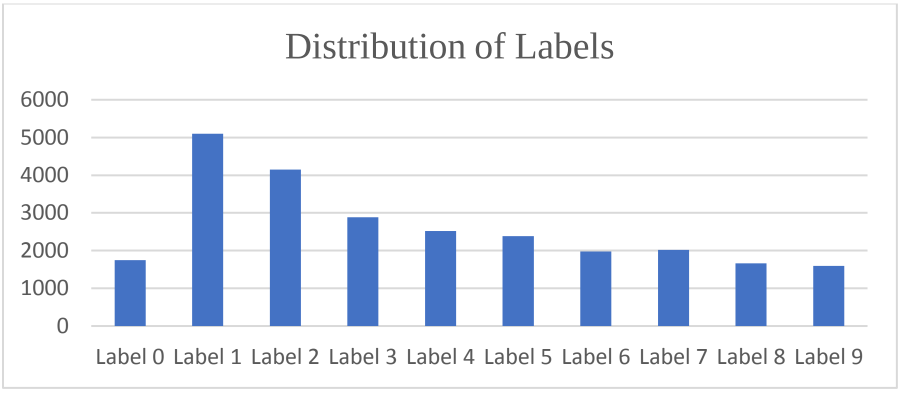

Figure 5. Distribution of labels

Metrics

To evaluate our loss, we've measured the average over cross entropy for all output labels. In other words, because we have multiple outputs, we measure cross entropy over each output, then calculate their mean. Equation 1.0 explains the loss function. (*n=number of outputs, m=number of classes, p_i = probability of target label, q_i = probability of predicted label)*

$$\text{Loss} = \frac{1}{n}\sum_{0}^{n}\left(\frac{-1}{m}\sum_{i=0}^{m} p_i \log q_i\right)$$

Methods

Our approach is based on maximizing the probability of each digit's occurrence and the length of the number. We call our model M and the input X, so our goal is to maximize the log(M|X) in which M equals to set of random variable m_1 to m_5 where m_n is associated to digit of nth location and the random variable L which represents the probability of the length.

To achieve a maximum probability, we need to minimize the loss function. For this purpose, we have used "adaptive gradient algorithm" (AdaGrad) which is an optimized stochastic gradient decent algorithm. This algorithm increases the learning rate for more sparse parameters and decrease the learning rate for less sparse parameters. This is one of the reasons of its great performance in image recognition problems. [3]

To find out the probability distribution of each digit, we have implemented a Convolutional Neural Network with an AdaGrad algorithm to extract the feature variable (H). By getting the feature variable then we were able to train the model by back propagating on each individual softmax classifier. In other words, each digit will have an independent softmax classifier and if each of these digits does not exist, then that specific output will not backprop at all. Figure 6 depicts this algorithm workflow.

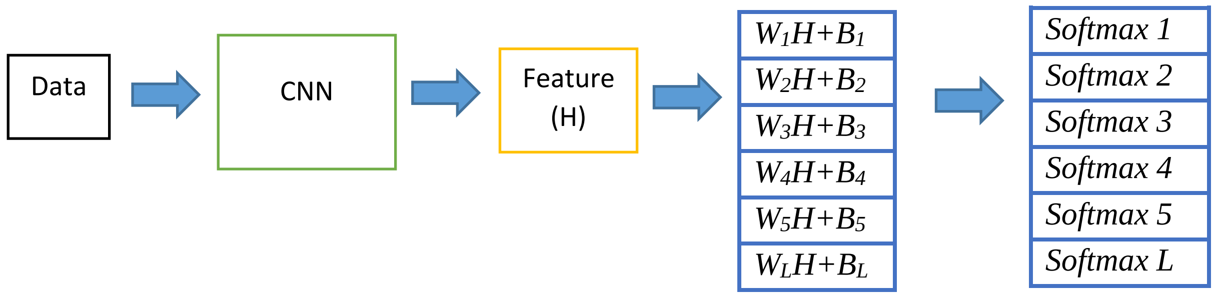

Figure 6. H: feature variable W: weight vector B: bias vector

Architecture

Convolutional Neural Network is the core algorithm utilized in this model. For this problem we have used 2 convolutional layer, In the first layer there are 8 kernels and in the second layer 16 kernels are implemented. After preprocessing the images, we feed the network with an input vector of size 784 which contains all pixels of a 28x28 sized image. When the input has been processed by the first convolutional layer, it will pass to an activation function which hear is a rectified linear unit, the result will go to a pooling layer with a 2x2 window size. This layer will reduce the size of the input in half. The same process will happen in the second layer. The only difference is that in this layer we have an input of size 14x14 and 8 channels. In second ConvNet layer as mentioned before we've used 16 kernels. Output of last pooling layer will go to a dropout layer which in our experiment, its dropping rate is 0.5. After dropping half of the units in our network, we finally feed the leftover units to six different fully connected layers. Figure 7 demonstrate the overall architecture of the model.

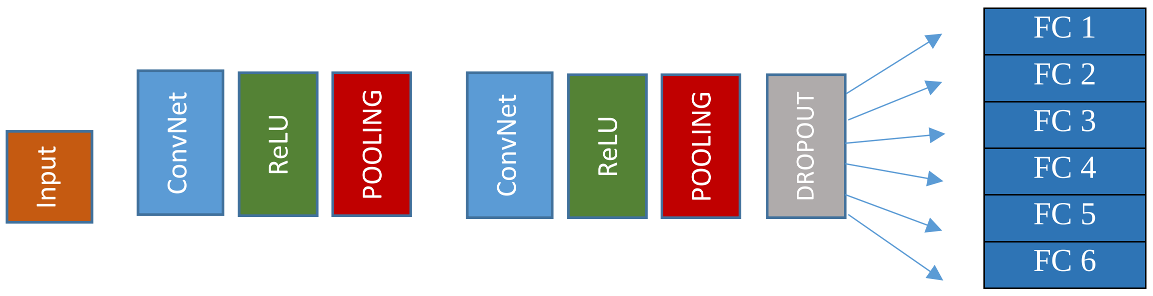

Figure 7. Model Architecture

First Convolution layer has k=8 and second one has k=16

The output of each fully connected layer will finally pass to a softmax function to get a probability distribution of each predicted digit and the length of the whole number. So basically we have seven independent output which their probability distributions are isolated and do not affect other outputs. We should imply again that if the target label distribution was all 0 values then that means no digit exists in that location, therefore, it doesn't impact the backpropagation process. Figure 8 shows how each fully connected layer is passing its output to an softmax classifier which will end up with the final result.

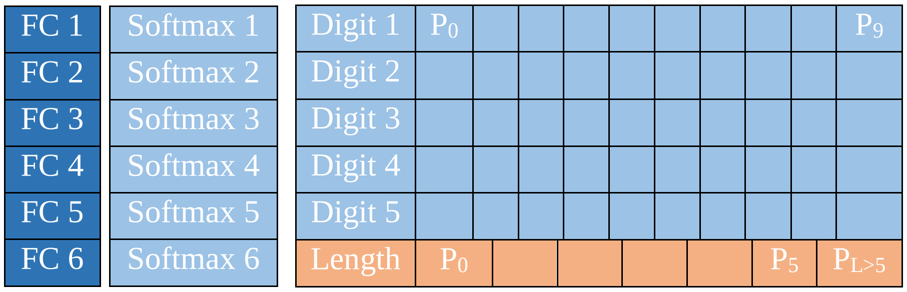

Figure 8. P_i = probability of the i^{th} class (class={0,1,2 … 8,9})

Let's have an inner look on convolution layer. In this model there are totally 2 convolution layers, first layer has a kernel size of 8 and second layer has a kernel size of 16. The patch size has been set to 5 with stride of 1 and same padding which means each image will be padded with zeroes to keep the image size same as the input. In figure 9 you can see a big picture of each convolution layer.

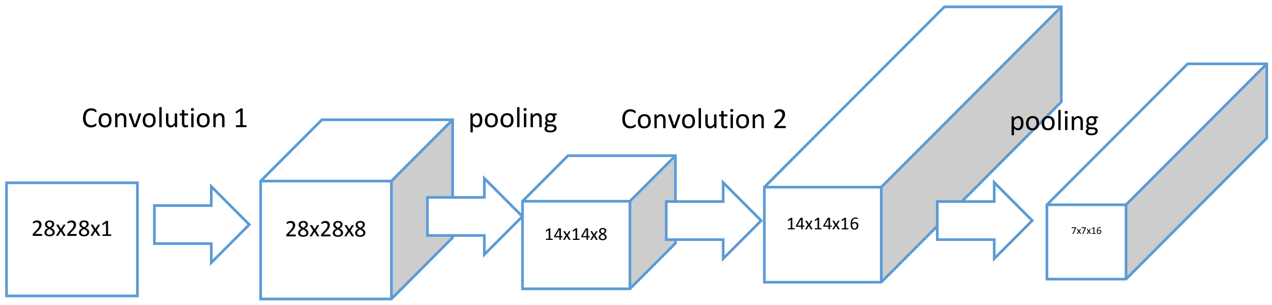

Figure 9. Convolutional and pooling layers

Now let's have a look on how actually this network perform on each filter with an example. We have used the same image in the introduction section to see how each filter is being activated. Figure 10 and 11 are filters output in layer one and two respectively:

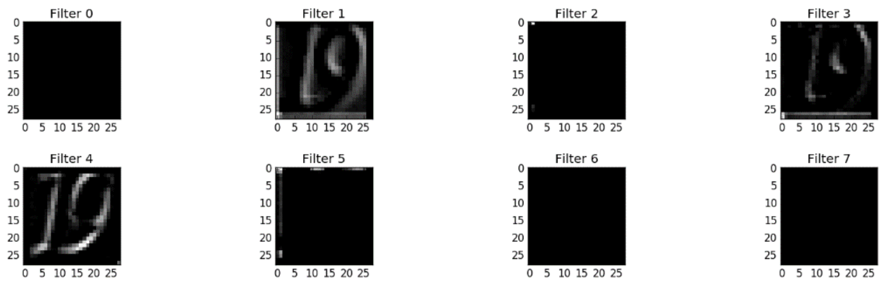

Figure 10. Filter activation in the first convolutional layer

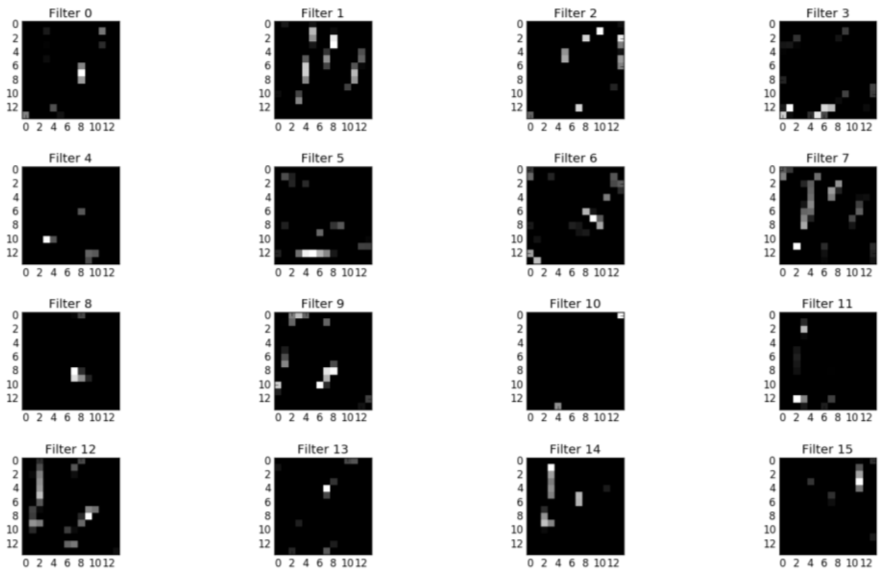

Figure 11. Filter activation in the second convolutional layer

Benchmark

Our goal for this experiment is to get an accurate model to predict the sequence of digits in a photograph. Here, accuracy refers to models ability to predict each digit's existence in the right place and also the length of the number. Human being's accuracy in catching the numbers correctly is about 98% therefore getting closer accuracy to this limit is the final goal of this problem.

Results

By running the training step for over 200,000 times of batch size 128 we have gotten an accuracy about 89%. As you can see in chart 1, the loss value has dropped from value 10 to 2.8 during the training process.

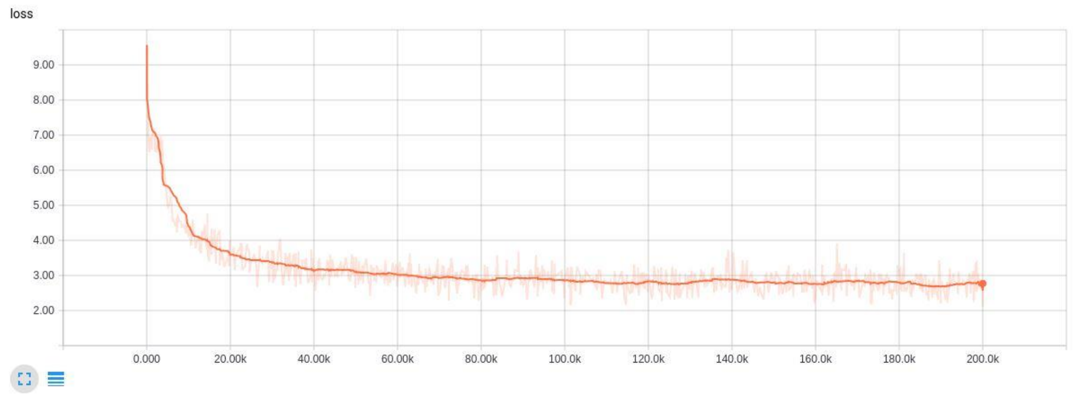

Chart 1. Loss Value

And you can see in chart 2 that test accuracy increase from 65% in step 500 to almost 98% in step 200,000.

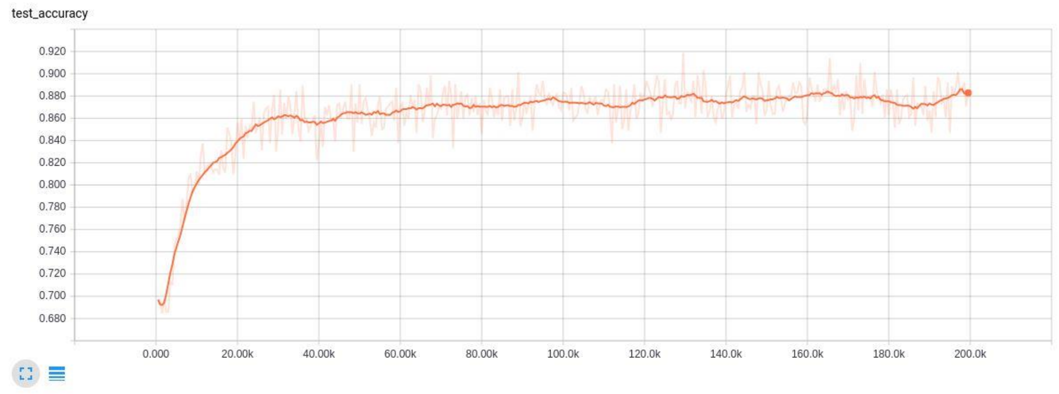

Chart 2. Test accuracy

Conclusion

Although, catching digits from a photograph is a complicated process but by utilizing convolutional neural network we can overcome this complexity and get a pretty accurate model. In this experiment we were able to get an accuracy of 89% of the test data. We can also get a better result by increasing the training time and the training data.

References

[1] Goodfellow, I., et al. Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks. ICLR, 2014 Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, Vinay Shet Google Inc., Mountain View, CA

[2] Street View House Numbers (SVHN): A large-scale dataset of house numbers in Google Street View images.

[3] Duchi, John; Hazan, Elad; Singer, Yoram (2011). "Adaptive subgradient methods for online learning and stochastic optimization" (PDF). JMLR. 12: 2121–2159.