Capstone Project
Machine Learning Engineer Nanodegree

Raman Khenanisho
November 1st, 2016

Definition

Project Overview

To create an accurate map, finding a way to recognize houses number in photographs is essential. Although the problem's 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].

Problem Statement

The goal of this project is to find a way to extract existing digits in an image in the right order. To reach this goal we need to utilize a learning algorithm which is able to learn the pattern of digits by watching multiple images. In our 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. Therefore, 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. To feed this algorithm we used SVHN (Street View House Numbers) data sets which is a real world image dataset obtained from google street view. This dataset has been used in the experiment mentioned in "project overview' section. We input the network with this dataset 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

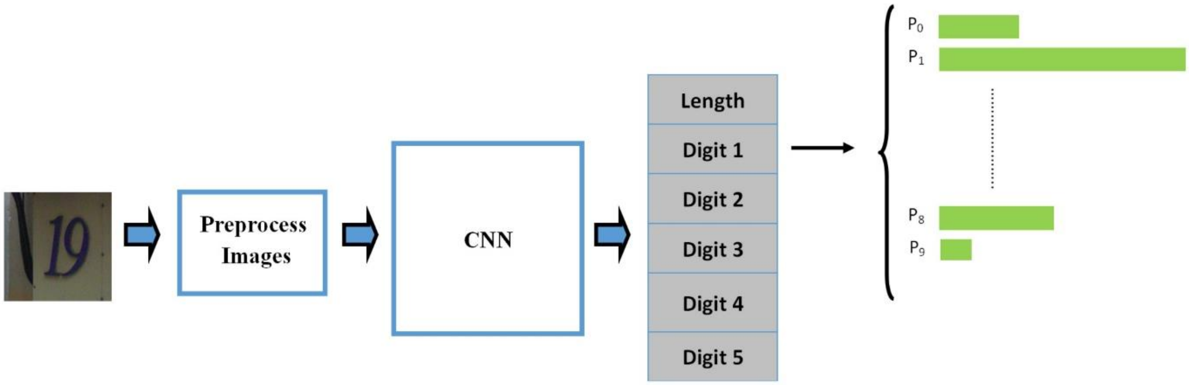


Figure 1. flowchart of the model

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

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)*

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

However, to compare our results from different runs, we used *weighted F1 score* because it provides more accurate result for imbalanced datasets as we will see later in data exploration section that our dataset is not balanced. And because we have multiple outputs then we need to calculate the average over F1 scores over all outputs prediction. So to sum up, our performance measure is as follow:

F1_avg = (F1(label_1_predicton) + …. + F1(label_5_prediction))/5

Analysis

Data Exploration

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.

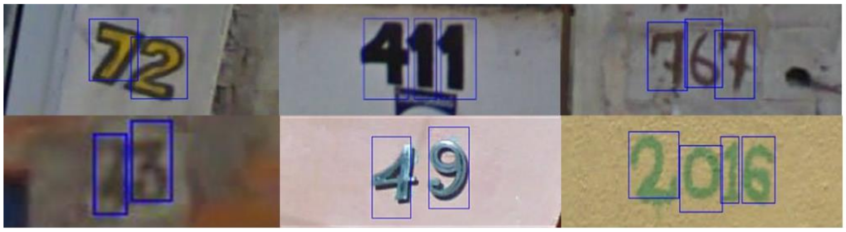


Figure 2. Sample set of pictures form training dataset

Although this data set is well formed to meet the specification of the problem but there are still some abnormalities and outliers to consider for example in Figure 3, number 8 in first picture is very blurry. Location of the number, size of the picture and the order of the digits are other issues to consider in this problem.

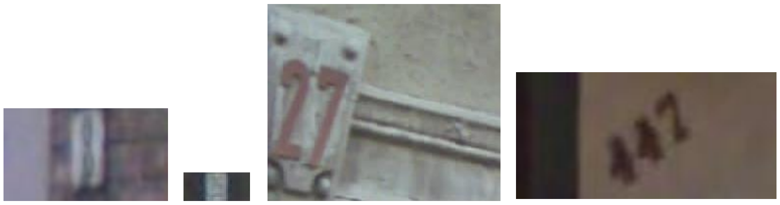


Figure 3. Sample set of pictures form training dataset

Image 1: very blurry   image 2: very small

Image 3: not centered   Image 4: not ordered horizontally

Exploratory Visualization

Figure 4 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. However, let’s see the average intensity value over the entire training data. Figure 5 explains how our data are separated in terms of intensity values. The standard deviation of this data set is *44.18*

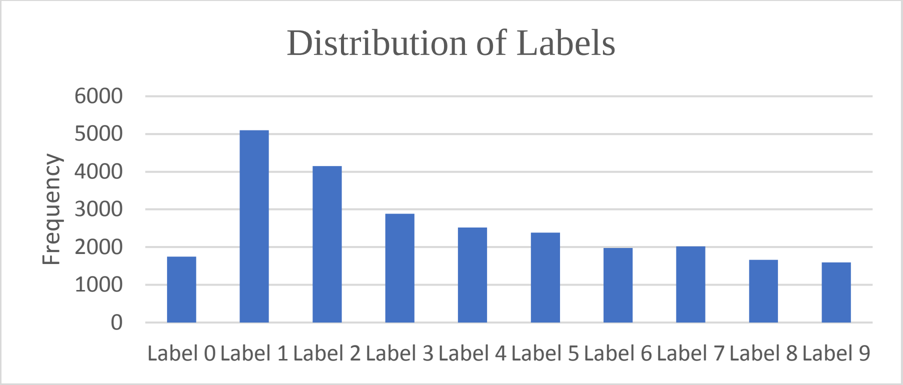


Figure 4. Distribution of labels

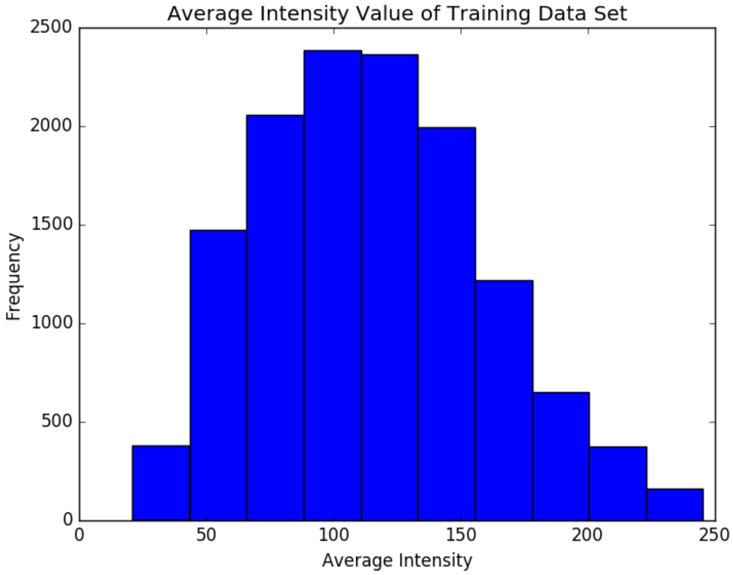


Figure 5. Histogram of average intensity of values in training data

Algorithms and Techniques

CNN Architecture

Convolutional neural network (CNN) is a feed-forward neural network which is inspired by animal's visual cortex. In simple words animals vision system is a connected layers of neurons that each neuron is responding to a specific region of space called receptive field. But all these neurons have an overlapping area with other neurons. The final image is coming by merging all these small activated neurons and output the corresponding picture.

The same process happens in ConvNets in essence that when an image is fed to the networks, it will be processed in a convolutional layer by multiple receptive fields which in this paper we call theme filters. Each filter is a square window with a side size equal to *patch.* We also need to define that how much should filters overlap; we call these distance *stride.* For example, in figure 6 we have a 5x5 input which is padded with zeros, a filter size of 3x3 and stride of 2. The orange part is the region of input that will be affected by filters in the first stride. The second strides region is denoted by the blue box. Therefore, we will totally have three strides horizontally and three strides vertically that gives us a 3x3 output (green box). The dot product of input vector and filters will be calculated and the value summed with bias is the output of that filter.

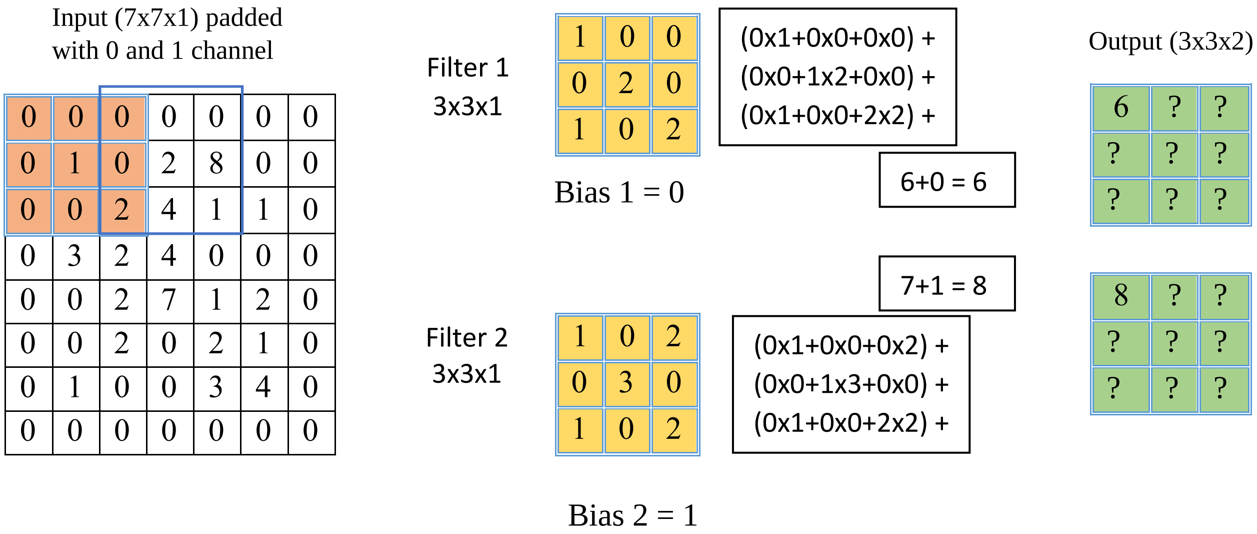


Figure 6. how filters work in ConvNets

Now that we understand how filters work in CNN, it's time to see what happens when we get their output. Let's assume that our input is a 28x28 image, and we have k=8 filters in the convolutional layer. We pad the image with zeros (also called *same padding)* to get the same size output as input. And our stride is 1 and patch size is 5, the one channel input will change to k channel output. Then the output goes to an activation function like ReLU (rectified linear unit), this function defines each neurons behavior and is max (0, x) which means if neurons value is less than zero, make it zero and if not just keep the value. Then we also have a pooling layer which basically reduce the size of its input based on its definition. We have different type of pooling layer like max pooling, L2-norm pooling or average pooling. But we just explain the max pooling algorithm. Let's assume that we have a pooling layer with filter size of 2x2, and stride of 2 then it will slide on the input and choose the maximum value of the 2x2 neighborhood and choose that value as a representative of the four. So it will reduce the size of its input to ¼ of the original size. And finally we have a drop out layer which if its probability value is 0.5, then it will deactivate half of neurons randomly. This helps the model to rely less on the data which has been trained of. In "Implementation" section you can find more detailed information of how each layer is acting.

Why Convolutional Network?

In machine learning, every pixel of the image acts as a feature for analysis. For example, a 28x28x1 image has 784 features. If we want to analyze it by a regular neural network, we need to feed all 784 features to all networks unit which is very expensive in terms of time and space. And also if we shift the image only by one pixel, it will change the result completely. However, by using ConvNets we can overcome these limitations because we have a set of neurons that share weights and each unit covers a small region of the image then we don't need to be worry about image shifting or its dimensionality very much. The only time that every input feature is connected to all units is on the Fully-Connected layer which usually is the last layer of the network. All these being said ConvNets are performing better job in image processing [4] and natural language processing problems [5]

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 roughly about 98% [6] therefore getting closer accuracy to this limit is the ideal goal of this problem. However, this goal is just good enough to trust the model to perform in human level. In this case by replacing human operator with an automated model we will get the same accurate job in a faster and more efficient way and of course the more we get closer to 100% accurate performance the better our model is but at this moment the initial goal of our project is to get at least at human's level performance.

Methodology

Data Preprocessing

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 7 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

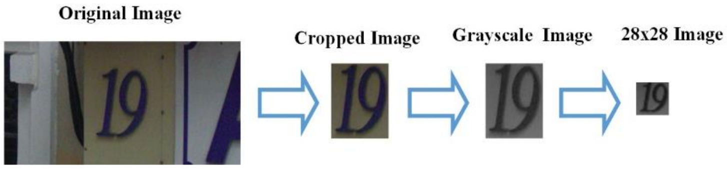


Figure 7. Flowchart of preprocessing images

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:

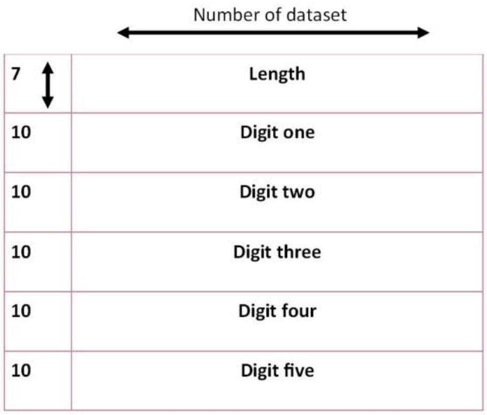
1) Digit Label

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 8. We saved these transformed data into 2 pickle files: data.pickle and test_data.pickle for train and test datasets respectively.



Number of dataset

7	**Length**
10	**Digit one**
10	**Digit two**
10	**Digit three**
10	**Digit four**
10	**Digit five**

Figure 8. final shape of data

Implementation

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 sparser 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 9 depicts this algorithm workflow.

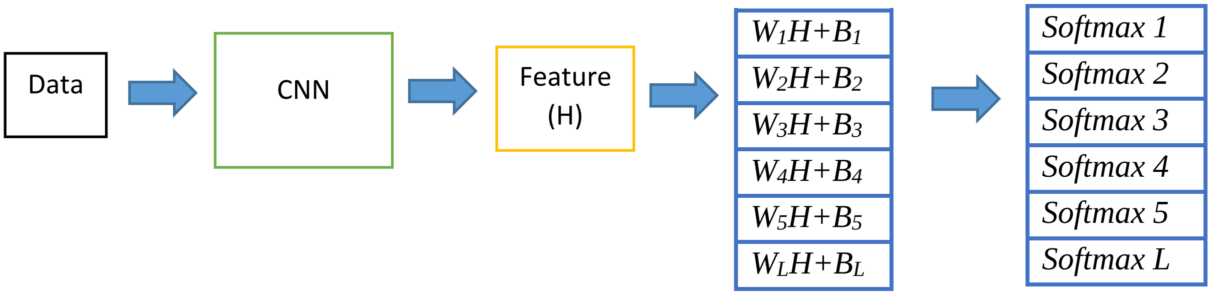


Figure 9. H: feature variable W: weight vector B: bias vector

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 10 demonstrate the overall architecture of the model.

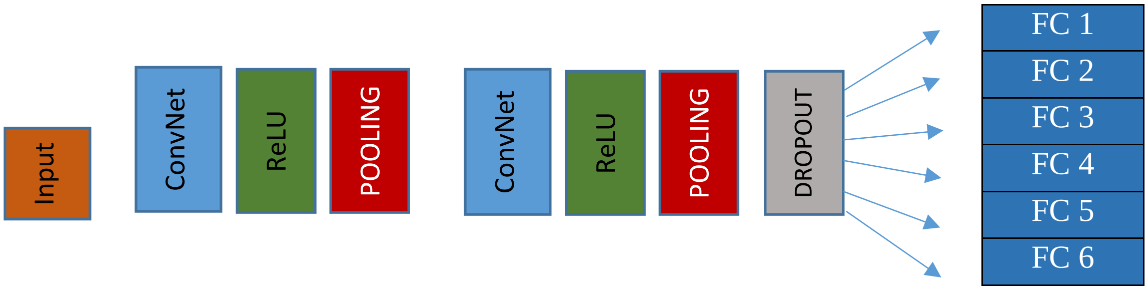
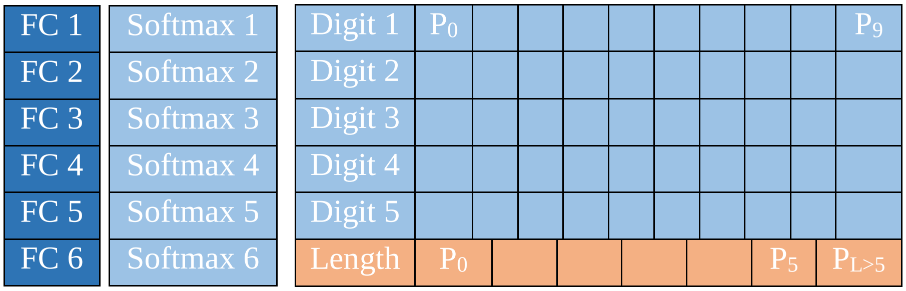


Figure 10. Model Architecture

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 11 shows how each fully connected layer is passing its output to an softmax classifier which will end up with the final result.



FC 1	Softmax 1	Digit 1	P_0									P_9
FC 2	Softmax 2	Digit 2										
FC 3	Softmax 3	Digit 3										
FC 4	Softmax 4	Digit 4										
FC 5	Softmax 5	Digit 5										
FC 6	Softmax 6	Length	P_0					P_5	$P_{L>5}$			

Figure 11. 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 12 you can see a big picture of each convolution layer.

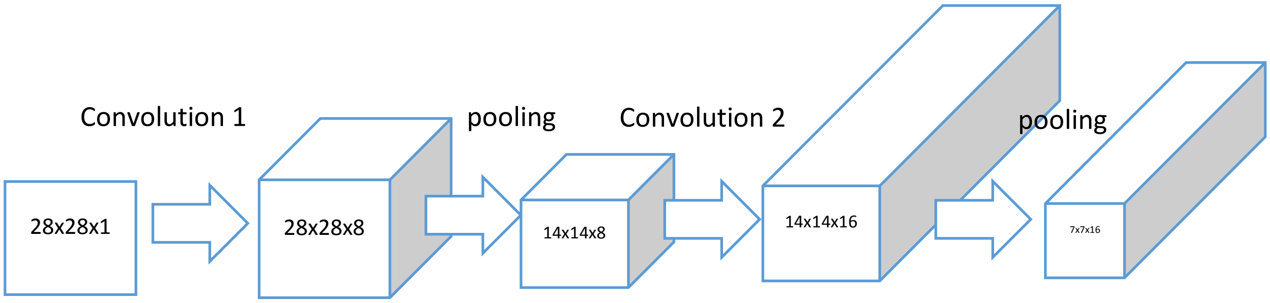


Figure 12. Convolutional and pooling layers

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 13 and 14 are filters output in layer one and two respectively:

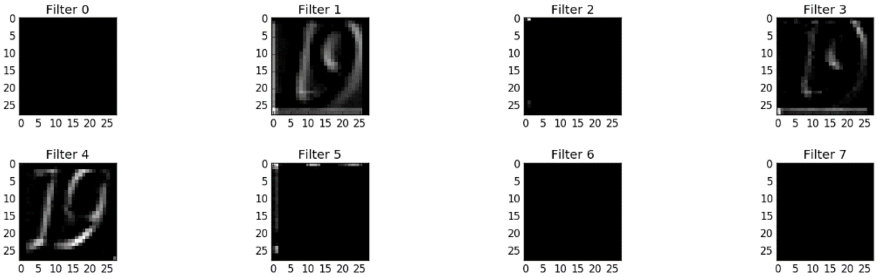


Figure 13. Filter activation in the first convolutional layer

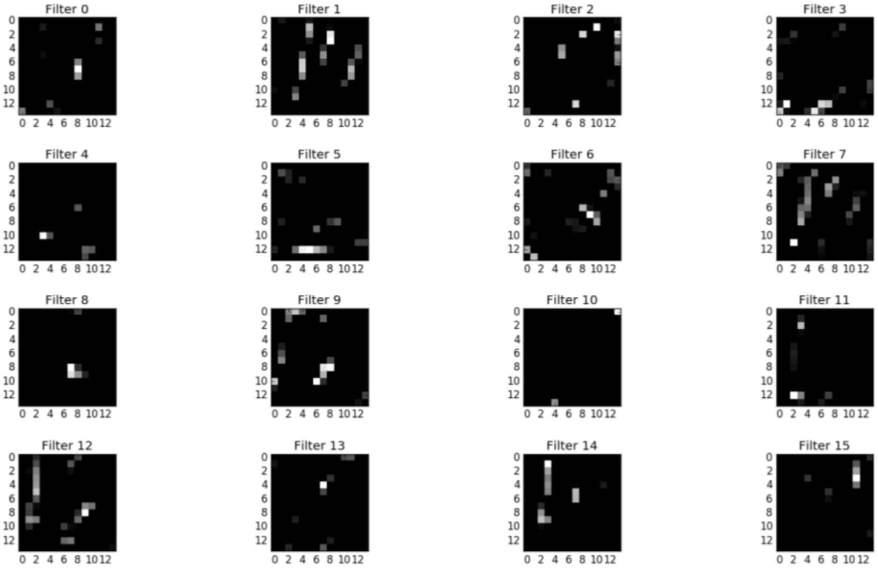


Figure 14. Filter activation in the second convolutional layer

Challenges

One of the challenges was to find a way to get 5 independent outputs with the shared feature input which was gotten from convolution network. But we were able to solve this problem but using 6 different fully connected layers.

The other issue was to generalize loss minimization to all of the outputs. This issue was solved by utilizing Tensorflows amazing function “tf.nn.sparse_softmax_cross_entropy_with_logits” which is able to calculate cross entropy toward multiple logits and at the same time apply Softmax to it. Then we simply were able to get the average of theme.

Refinement

In this experiment first we start with 4 filters in the first convolutional layer and 8 filters in second one but by changing the filters to 8 and 16 respectively we were able to get about 4% more accurate result. Then by choosing image sizes from 32 to 28 we were able to get a faster trained model with almost the same accuracy.

Results

Model Evaluation and Validation

An accurate model requires that we choose best parameters. We tried different setting of batch sizes and dropout rates to see the model’s convergence in the first 5000 runs and based on the results we choose the right parameters. In Table 1 you can find the models performance under different parameters.

Drop Rate	Batch	Step 1000	Step 2000	Step 3000	Step 4000	Step 5000
0.5	64	0.72	0.69	0.75	0.74	0.74
0.5	128	0.71	0.71	0.75	0.75	0.78
0.7	64	0.69	0.66	0.71	0.73	0.73
0.7	128	0.69	0.68	0.69	0.72	0.72
1.0	64	0.66	0.70	0.69	0.70	0.70
1.0	128	0.69	0.68	0.68	0.68	0.66

Table 1. F1-score for different parameters

As you can see the best convergence is happening on dropout rate 0.5 and batch size 128. So we trained the modeled with these parameters and in average we were able to get 12% more accurate result compared to drop=1 and batch size= 128. Finally, we trained the model for 200,000 steps and get a F1-score about 89%.

Justification

Our final result was F1-Score equal to 89%. Although, our benchmark which represents a human agent accuracy is 98% but if we consider the limited dataset which had less than 40,000 pictures and also the limit of resources that would explain the 9% less accurate model. The best performance so far was achieved by "*ICLR, 2014 Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, Vinay Shet Google Inc.*" [1]. They were able to get 97.84% accuracy.

Conclusion

Free-Form Visualization

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. The interesting fact is that the difference of loss from step 40,000 to 200,000 is about 0.3 which although is notable in loss reduction but shows that after step 40,000 the slope of loss reduction is decreased significantly which means that network has already learned most of the patterns in the dataset and is updating the weights less than previous steps.

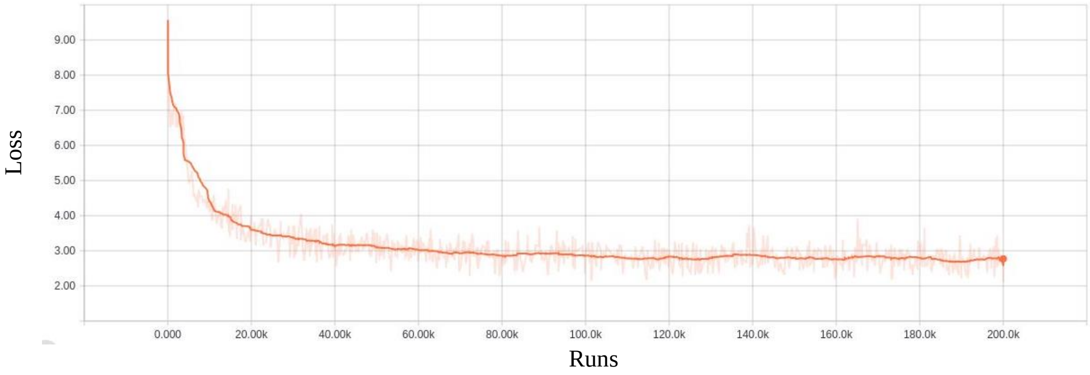


Chart 1. Loss Value

When model was trained, we've tested the model over 2000 steps with a batch of 128 pictures each time (test images are unique and model has never seen theme). Chart 2 shows the histogram of results and you can see that in almost 1600 runs we have gotten an F1 score more than 86% and the mode of the results is about 87%. Then we can conclude that our model is about 87% to 89% accurate.

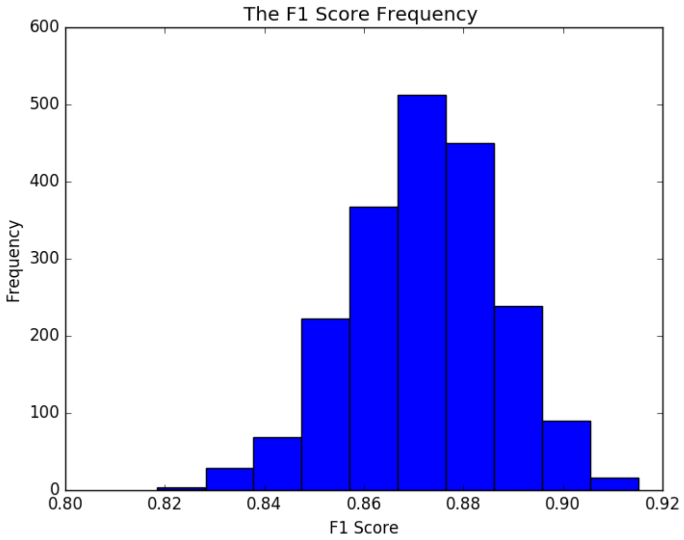


Chart 2. Test accuracy

Reflection

In this project we tried to train a model which is able to identify number in photographs. To do so we used the SVHN dataset which is obtained from google street view pictures. We preprocessed the dataset to minimize the cost of time and memory. Then we utilized the convolutional neural network as a training approach. However, this was not an easy task to accomplish since we had to find a way to identify all digits at the same time which means we must have multiple outputs and because our outputs are simply a probability distribution of any digits' occurrences we had to find the length of the number. The solution that we came up with was simple we just needed to add more fully connected layer at the end of the network and each output had its own weights to get updated. Therefore, we just needed to extract the features from input and share it with different softmax classifiers. Finally, when the model was ready we tried different hyper parameters like dropout rate and also batch size to find the most efficient wat for training model. After finding the right parameters, then we trained the model for over 200,000 runs and got an amazing 89% F1-score which maybe doesn't seem very good at the first look but for this limited data, this model did a fabulous job.

Improvement

Although we got an accuracy about 89% in this model, but this does not satisfy us. We believe that by using more input data and also keeping all three channels of the pictures we would be able to reach even more accurate model. For next step we might use an extra 531131 pictures which are also available in SVHN data set. And keep the three channel of the pictures. However, this will add on memory and time cost which need more physical resources and stronger GPUs.

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.

[4] LeCun, Yann. "LeNet-5, convolutional neural networks".

[5] Collobert, Ronan; Weston, Jason (2008-01-01). "A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning"

[6] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, Vinay Shet (Submitted on 20 Dec 2013 (v1), last revised 14 Apr 2014 (this version, v4))