Capstone Project

Machine Learning Engineer Nanodegree

Gene Der Su
September 30st, 2016

I. Definition

Project Overview

The objective of the project is to build an app that can interpret numbers in real-world images. I have trained a model that can decode sequences of digits from natural images, and wrote a Python script to generate the location of detected digits as well as classify which digit they are. The task can be breakdown into two steps. First, identify numbers in the image with their boundary boxes. Second, classify the image in the boundary box to find out the actual digit.

The street view house numbers (SVHN) dataset is a good dataset to start with. It was obtained from house numbers in Google Street View images with a large amount of labeled data. SVHN contains both the boundary boxes of the digits and the actual class of the number. Therefore I can easily train the first and the second step within the same dataset, as how it would be tested on real-world images.

This project can be farther applies to other usages in many machine learning areas. Just by replacing the dataset, it can be used to detect the pedestrian on the street for self-driving cars, finding criminals in the security camera footages, and identify items in an image for shopping queries. The usage is unlimited, and it is a great project to learn about deep learning and applications to real-world problems.

Problem Statement

The problem is to identify all digits in a real-world image and classify each digit in a timely manner with good accuracy. It cannot be framed as a simple number classification problem because the number space will blow up (i.e. all real

numbers are unlimited). It also cannot be framed as a localization problem because we do not know how many numbers is in an image. Moreover, there will be a need to train one model for each length of digits and needing more data to support different numbers of digits. The best way to solve this problem is to first find all the digits in the image, then for each identified region, classify which number is the digit. We call this problem object detection problem.

Researchers have been working on the detection problem for decades. The most recurrent methods I found were YOLO and faster r-cnn. Both methods give real time results with high accuracies. However, they both require NVidia GPU's to really take the advantages of the speedups and they don't currently supported with Google's TensorFlow. Therefore, I chose to use a simpler approach of combining a region proposal with a classification net. I first use OverFeat to propose the region of interest, then for each region of interest I used a simple classification neural net to find the digit.

Metrics

The desired model should come with high accuracy in the least amount of time possible. It should find all the numbers in a given image with the correct label. However, randomly generate region proposals should cumulate some panties. Therefore, the accuracy will be calculated as the proportion of identified digits, multiples the proportion of correct proposal, multiples the proportion of correct class given the proposal is located correct. The time will simply be the time used for both detection and classification stage.

Since the model should really focus on real-world application, it should be evaluated on real-world images. I will go out and take few pictures around the neighborhood and calculate the performance matrices based on those images.

II. Analysis

Data Exploration

SVHN is a real-world dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. It can be seen as similar in flavor to MNIST (e.g., the images are of

small cropped digits), but incorporates an order of magnitude more labeled data (over 600,000 digit images) and comes from a significantly harder, unsolved, real world problem (recognizing digits and numbers in natural scene image). SVHN is obtained from house number in Google Street View images.

Following are some examples of the boundary boxes drawn on the image:

Dataset counts

	Image count	Total digits count	Average digit per image	SD digits per image
Test	13068	26032	1.992	0.0340
Train	33402	73257	2.193	0.0288

Average sizes

	Average image width	Average image height	Average digit width	Average digit height
Test	172.583	71.566	30.909	55.701
Train	128.285	57.213	36.518	74.264

Number of digits in an image

	1	2	3	4	5	6
Test	2483	8356	2081	146	2	0
Train	5137	18130	8691	1434	9	1

Class Count

	0	1	2	3	4	5	6	7	8	9
Test	1744	5099	4149	2882	25384	1977	2019	1660	1585	1744
Train	4948	13861	10585	8497	7458	6882	5727	5595	5045	4659

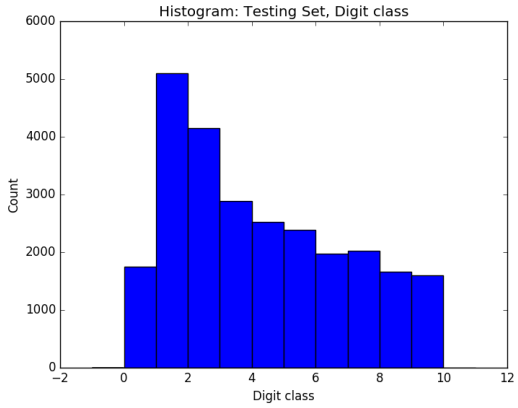
There are some characteristics to point out in the statistics above. First, the training set is close to three times of testing set and they have similar statistics for their average number of digits per images and the standard deviation of the number of digits per images. However, the image sizing and the digit sizing for training set and testing set are quite different. This points to a potential preprocessing stage to standardize the size of images and digits before feeding them into the networks. When looking at the number of digits count, both training and testing set doesn't have many examples for images with more digits. This may lead to the trained model unable to detect images with a lot of numbers.

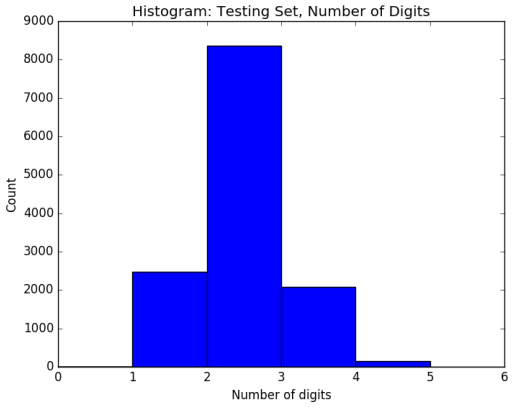
When looking at the class count, its obvious there are some number appears much more frequent than the others. This can lead to the network classifies one number more frequent than the other. Also, just looking at the raw labels, I found a few images that had incorrect labels for the boxes. This can also lead to networks not able to find the correct pattern or taking longer to train.

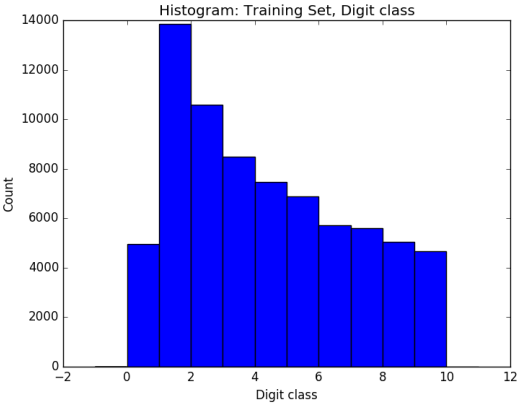
Exploratory Visualization

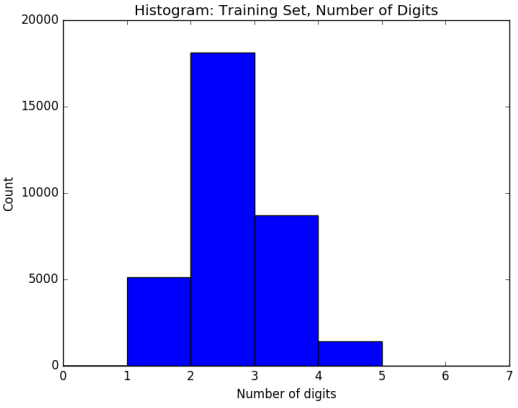
The first two histograms are the frequency of digit class and the number of digits for the testing set. The next two histograms are the frequency of digit class and the number of digits for the training set. We can easily note that the number between 1 and 5 has much higher frequencies and the rest of number have around the same frequency. This frequency difference might cause the classification model to classify 1 to 5 more frequent then the rest of numbers and cause some degree of error in the evaluation. Although both training set and testing set have similar distribution, the histogram is saying length of two appears

much more frequent and its not possible to have a really long digits in the image. This might also be a problem in the detection part of the networks thinking that there will only be limited amount of digits in the image.









Algorithms and Techniques

I first started to solve the problem by training a network for the length of the digits then use that information to classify n times for each of the numbers. However, no matter how I tune the model, it never had an accuracy of more than 65 percent. Thus, I did more research on how other people solve this kind of problem. I came across a method called YOLO that trains the location of the boxes, the confidence of the box, and the probabilities of the classes altogether in a network. I attempted to use their setup to train on the SVHN dataset. Not only its time consuming on CPU (I don't have a GPU) but I also got a divergent result. I then changed my focus on to finding the individual digits region first, and then use that information to do the classification. I found out someone had already implement a method called OverFeat in TensorFlow and that I can use it

directly in my project for the digit detection. Then I built my own networks for the classification of single digit and put the two models together. The result looks great!

Benchmark

I will benchmark on the real-world images I got for the testing purpose. There are 21 images total. The performance will be calculated based on the definition in the matrices section.

First, the average time per image I got on my computer for evaluation these 21 images on a 2015 MacBook Pro 15" with 2.8GHz Intel core i7 is `4.3907551879` seconds/images. The accuracies looks like the following:

	Proportion of identified digits	Proportion of correct proposal	Proportion of correct class given the proposal is located correct	Accuracy
Image 0	2/2	2/2	2/2	1
Image 1	4/4	4/4	4/4	1
Image 2	4/4	4/4	4/4	1
Image 3	4/4	4/4	4/4	1
Image 4	4/4	4/4	4/4	1
Image 5	3/5	3/6	3/3	0.3
Image 6	3/3	3/5	2/3	0.4
Image 7	4/4	4/4	4/4	1
Image 8	0/2	1	1	0
Image 9	4/4	4/4	4/4	1
Image 10	2/2	2/4	2/2	0.5
Image 11	5/5	5/6	5/5	0.833

Image 12	3/5	3/3	3/3	0.6
Image 13	1/1	1/2	1/1	0.5
Image 14	2/2	2/6	2/2	0.333
Image 15	5/10	5/6	5/5	0.4167
Image 16	3/5	3/6	3/3	0.3
Image 17	3/3	3/3	3/3	1
Image 18	2/3	2/2	2/2	0.667
Image 19	3/3	3/3	3/3	1
Image 20	4/4	4/4	4/4	1
Average	0.8556	0.8381	0.9841	0.7071

III. Methodology

Data Preprocessing

There are a lot of works put into data processing. First, the original labels provided in SVHN data are in the form of Matlab struct. I have tried using python to directly reading those .mat, but python was not very efficient on reading them. I then directly write a script in Matlab to go through the struct and save the labels into a .txt file. Therefore python can easily parse the .txt file into the necessary information for the part of training.

Second part of the data preprocessing comes from converting these txt file labels into the format that Tensorbox takes. Going through their example, both training images and testing images need to be converted into 640 by 480. There are also idl file and json file for the labels in both dataset that need to be generated. I followed the required format and wrote a python script to generate the required format for Tensorbox training.

Lastly, the training of my own classification network will be much convenient if I can read the data directly from a pickle file. I used similar a design as in the one

in Udacity deep learning course. I cropped each digit and resized into 28 by 28-single channeled image and normalized them with zero mean and one standard deviation. As for class label, I directly append the integer number onto a Python list. In the end, I pickled a tuple of the image pixel data and the labels, which both of them are stored in a list. The image pixels are stored as ndarray in the image list.

To solve the problem of different image and digit size, the images are rescaled to 640 by 480 and the digits are rescaled to 28 by 28. The problem of unbalanced classes for each digit is actually not a big problem after I see the result of the classification network being 98.41% accurate. It might be due to the natural occurrence of the street numbers not being uniform. As long as the testing set follows the same distribution, the model should work just as well. Another possibilities might be that even though they are not uniformly distributed, all classes have enough training examples for the network to learn the pattern of the classes. Therefore, all classes are well trained even with a different amount of examples.

Implementation

The digit detection part of the network was implemented with GoogLeNet-OverFeat with Russell's TensorBox. The digit classification was implemented with three fully connected layers each with 1200, 300, and 75 nodes. They all included a 0.5 dropout rates. The learning rate was initialized at 0.3 and decay for every 5000 steps with 0.9 of the pervious value. The stochastic gradient descent has a batch size of 128. The checkpoint was set at every 1000 steps and the model was set to run a maximum of 200,001 steps before termination.

Refinement

My initial solution was using a convolutional neural network to detect the number of digits in the image then use a recurrent neural network approach to classify each digit. The accuracy was terrible, as no matter how I adjust the length prediction networks, the accuracy of the first networks would not pass 65 percent. Therefore, even if I have a perfect network to classify all digits with 100 percent accuracy, the overall accuracy will still be bounded by 65 percent and would consider a bad method.

The second approach I intended was to use YOLO for objective detection and classification at the same time. It predicts a set of boxes contains the object, the confidence of the boxes, and the probability of each class as its output. I can then put a threshold on the confidence and the probability to report only the most likely digits in the image. The method was implemented in darknet, so I had installed all the necessary dependences on my computer to run the framework. I successfully pre-processed SVHN dataset into the required format and run the model for a few days. However, unfortunately the model had diverged. I figured that the framework was running very slow on CPU and that it's not robust enough to take SVHN dataset.

In my last attempt, I found a robust method called OverFeat and that there is an implementation of the method in TensorFlow ready to use. So I did another data pre-processing for the Tensorbox and this time it works great. All the intermediate results showed very good boundaries that boxed around each digit. Although some images seen not have good labels, the overall result seems very good. Therefore I continued with implementing my own classification networks. It was a three fully connected layer neural network. Initially I was just thinking to test it out with such a small networks. However, the accuracy was easily gone above 90 percent, so I just fine-tune the model for the best result with the networks. The rest of work comes to putting two models together and produce the results from a given images.

IV. Results

Model Evaluation and Validation

This model was chosen because it solves the problem with good accuracy. I have generated 21 images showed in the *images_input* folder that are unseen to the model during training. This model gets an average accuracy of 0.8556 on detecting the digit, 0.8381 on making the right detection, and 0.9841 for the correct classification on this set. The model is robust on different kinds of font, color, and location for most of images. However, a few situations I found that the model is not doing so well are hand written digits (image 08), image with English characters (image 14), and very long number (image 15). In addition, if the digits are not big enough, the model might not be able to detect them. In general, it can

all be the result of the limited training data as the model learns to make the prediction within the dataset. If I have generate artificial images without these characteristics, the model might be able to do better in these special cases.

Justification

The benchmark results all seem really good. The model gets an average accuracy of 0.8556 on detecting the digit, 0.8381 on making the right detection, and 0.9841 for the correct classification on the benchmark dataset I provided. I would expect the model having an overall accuracy of less than 60 percent as YOLO only had 59.2 average precision on VOC 2007 dataset. However, the result of the model from my own dataset gets a better average accuracy of 0.7071. Overall, it's a great model to use in digit detection.

V. Conclusion

Free-Form Visualization

In this section, I will present the output images of the model evaluated on my own image set. You will see the good and the bad of the model such that it can detect digits with different font, size, and location and that it had a hard time detecting hand written digits, images with characters, and very long number. The detected digits will be boxed by white rectangle; the predicted class will be shown in the upper left corner of the box. You can zoom in the document to see the number more clearly.


1900
2414
2412
12
To Berkeley BART
BART
511
2 HOUR
PARKING
8AM-7PM
2223


25
1-800
9am to 6pm
9/25/16
9:56
SEP 28
2 hr
Max Parking
Premium Rate

Reflection

The solution used to interpret number strings in real-world images is solved by break down into two sub-problems, one that detects the digits in the image, and another that classifies each detected digit. The detection was solved with state of the art GoogLeNet-OverFeat using Russell's TensorBox framework. The classification of the digits was solved with simple three fully connected layer neural networks. The interesting thing is that the classification had a very good

accuracy with this small size of network. Without the need of making the model more complex, it gives us a simple model with high accuracy. The bottleneck of the overall accuracy lies on the detection part of the problem. However, even state of the art YOLO had less than 60 percent average precision on the VOC 2007 dataset, 70 percent overall accuracy on my model solving the digit detection problem feel very satisfying.

Improvement

The weaknesses of my model are hand written digits, images with characters, and long numbers. I think all of them appeared because of the dataset I trained on. In SVHN dataset, there are very little of such examples for the model to train. One possible solution to solve the problem is simply artificially generating the dataset with the property with these weaknesses. We can incorporate with the MNIST dataset with computer fonts at random location and size on some natural background images. We will also add random characters into the generated images to have the example of non-digits. We can vary the number of digits we insert into the image to have more examples for long numbers. That way all weaknesses will be covered and the model will improve its performance on such examples.

References

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. *arXiv.* May 9th, 2016.

[2] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *ArXiv*. January 6th, 2016.

[3] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading Digits in Natural Images with Unsupervised Feature Learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning.* 2011.

[4] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. *arXiv.* February 24th 2014.