Capstone Project

Machine Learning Engineer Nanodegree

Terence A. Cooper
August 16, 2016

Definition

Project overview

Could machine learn things like humans? While the question has long been asked thousands years ago, the answer is not clear until recent several decades, thanks to the persistent endeavor of scientist and researcher community, now, I think a lot of people are inclined to say machines can learn . (Do they, however, learn like humans? We can not answer it yet.) This project uses one type of the machine learning models called deep convolutional neural network to train a model to recognize arbitrary multi-digit numbers from Street View imagery[1][i]. Research of convolutional neural network at the earlier stage was around 1980s-1990s. Unfortunately, people don't have enough data to make it more popular like it is today, back then. The other factor that prevent it from more commonly used cases is the limitation of computational resources. It is worth noting that the very fact that revival of deep learning is brought back by 'Big DATA' and relatively cheap hardware, which means the surge of computational capacity. Today every one can easily get the data that 'big' enough and the computational resource in abundance for his/her intention. This is why this project can be completed.

Problem Statement

Ever since I have watched the movie <Zootopia>, after Nick the fox took the bunny to run a license plate, I was wondering why the service personnel eventually become 'sloth', that they move, speak, even laugh slowly. I guess one important reason is: they don't want to make mistakes in their work and the whole 'slow' thing could help that. Then I ask myself, how we could help them? If the bunny show the service personnel a picture, instead of telling him the number, and the computer itself can recognize the numbers in the picture--people who run the plate doesn't have to hit the keyboard themselves--this way they don't make any mistakes, hence they may be from 'slow' to 'quick'. So the problem is can a computer recognize multi-digit numbers in a picture of license plate? After reading the report I think you will have your answer.

Metrics

In the experiment, I used accuracy to measure the performance of the model.

$$\text{accuracy} = \frac{truepositive}{numberofdigits}$$

The numerator 'true positive' means the number of correct predictions per digit, and the denominator 'number of digits' references to the total number of the digits in current batch of data points. It is calculated by batch_size*number_of_digits_in_one_data_point.

Discussion of commonly used metrics:

In terms of each digit, what we really care about is whether the prediction is the same as the true label, this makes it a binary classification.

In the context of binary classification:

Accuracy - How many instances did the model label correctly?

Recall - How often was the model able to find positives?

Precision - How believable the model is when it says an instance is a positive?

Obviously, we only care about the model's accuracy here.

Analysis

Data Exploration

The dataset used in this project is The Street View House Numbers (SVHN) Dataset[2][ii]. SVHN is a real-world image 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 images). SVHN is obtained from house numbers in Google Street View images. Below are some of the example pictures from the dataset.

These are the original, variable-resolution, color house-number images with character level bounding boxes, as shown in the examples images above. (The blue bounding boxes here are just for illustration purposes. The bounding box information are stored in digitStruct.mat instead of drawn directly on the images in the dataset.) The whole dataset comprises three compressed files: train.tar.gz, test.tar.gz, extra.tar.gz and the total number of pictures is 248,823. The below dictionary structure is:

{label_length: number_of_this_kind_of_labels}

train dataset: {1: 5137, 2: 18130, 3: 8691, 4: 1434, 5: 9, 6: 1}

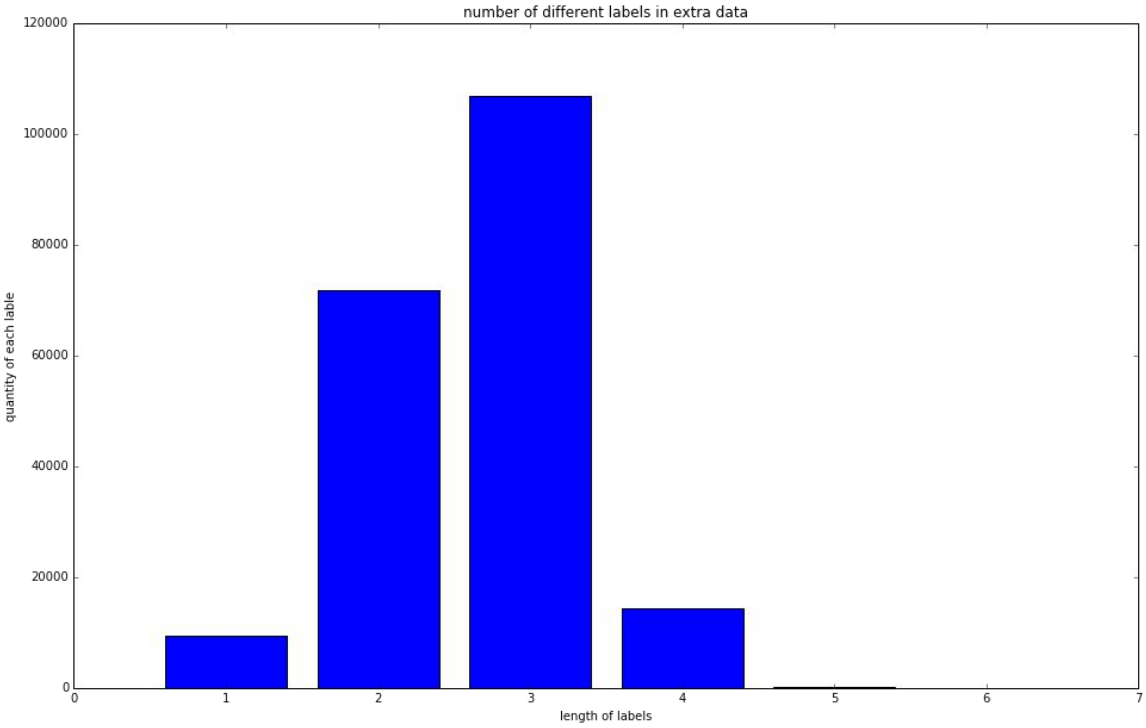
test dataset: {1: 2483, 2: 8356, 3: 2081, 4: 146, 5: 2, 6: 0}

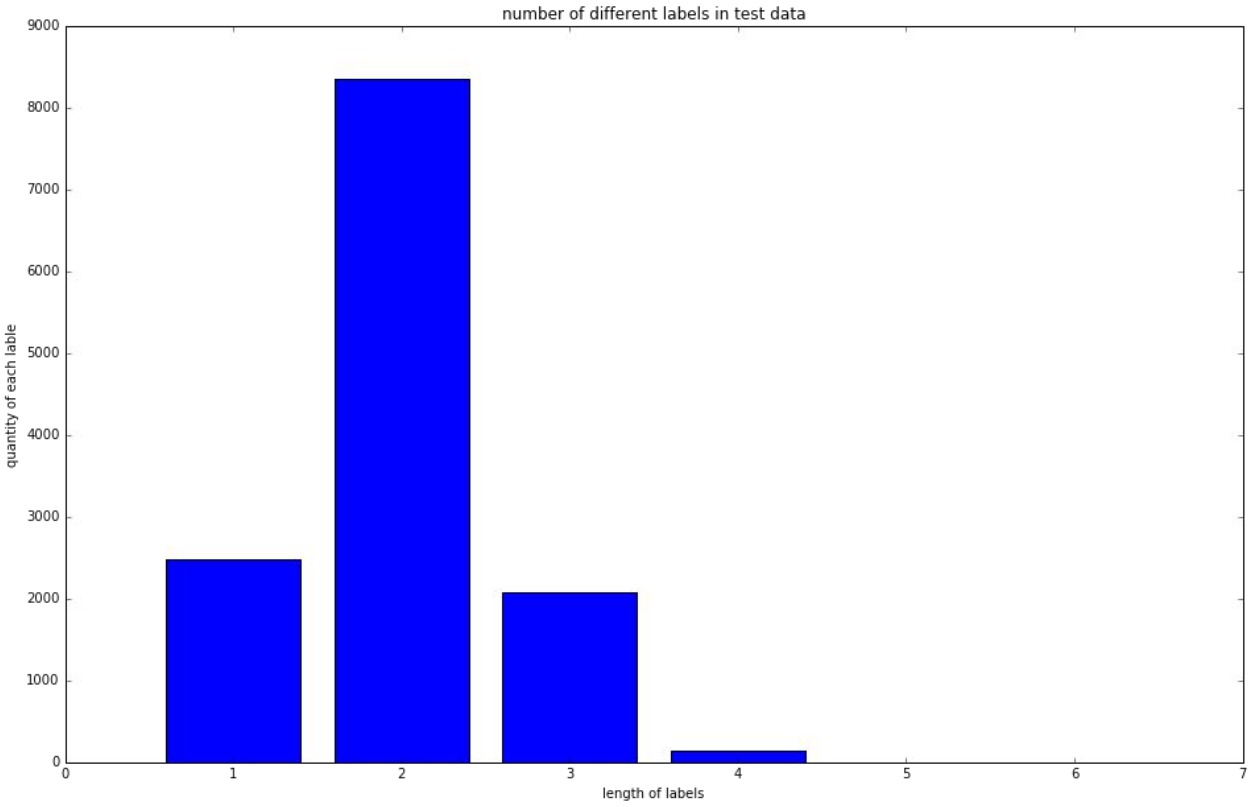
extra dataset: {1: 9385, 2: 71726, 3: 106789, 4: 14338, 5: 115, 6: 0}

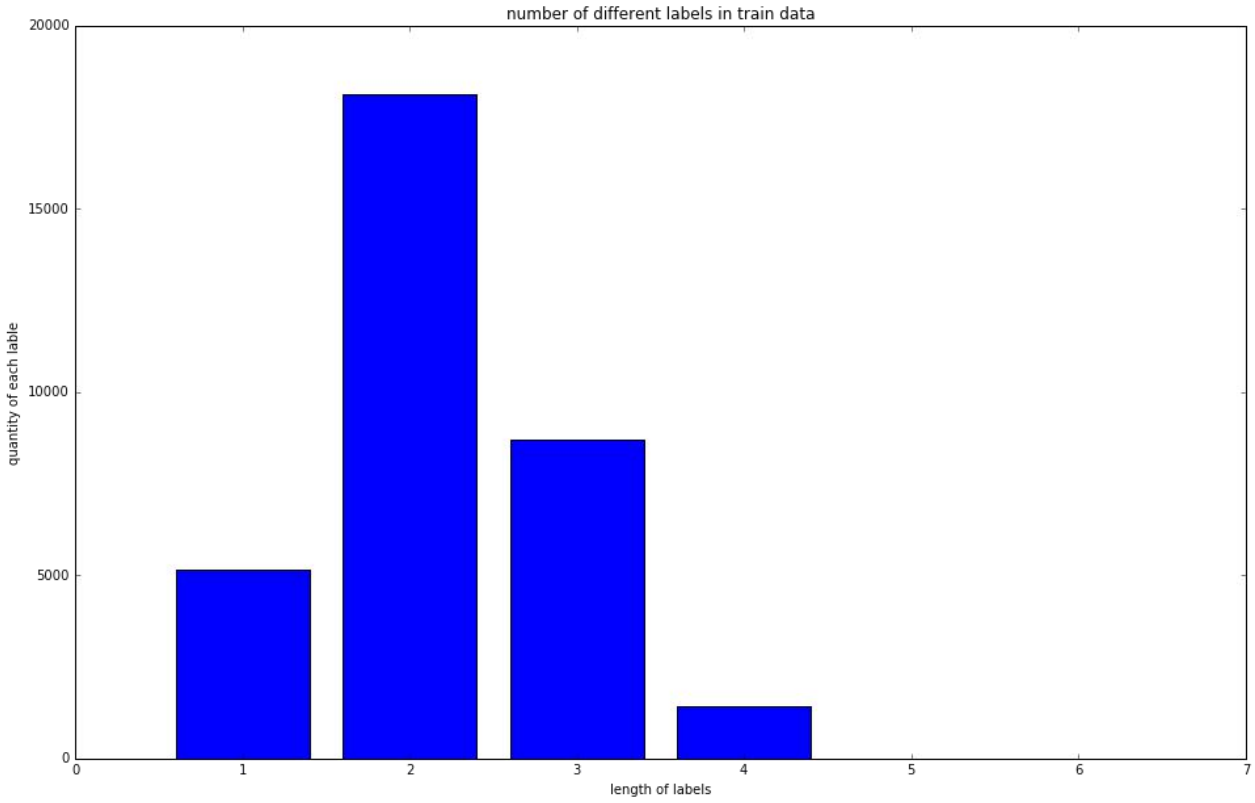
In the dataset, there is only 1 picture whose label length is more than 5, so we removed it from the training dataset.

Exploratory Visualization

Below are the distributions of labels in each dataset.







We can see the distributions of train data and test data are somewhat similar, which is what we expected. This way the test result is more of a representation of the train result.

Algorithms and Techniques

A Convolutional Neural Network was used to build the architecture of the classifier model. This network is the state-of-art algorithm for most image processing, including classification. It surely needs a large amount of data comparing with other approaches, fortunately the SVHN dataset is big enough for the training.

In the experiment, I used SGD(Stochastic Gradient Descent) as the main algorithm with learning rate decreasing over time steps. During the training process, we first randomly shuffled the whole training dataset; at every

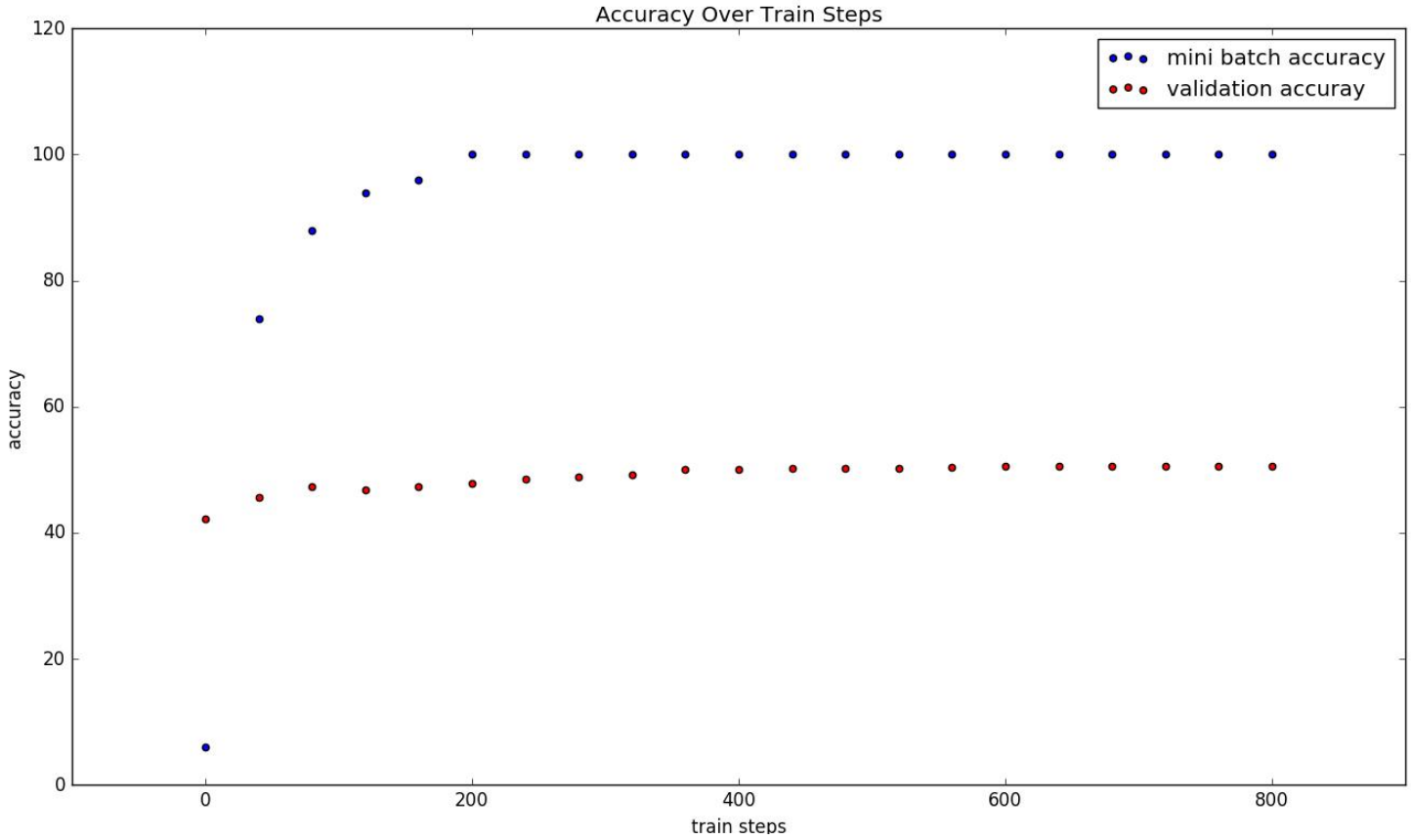
time step we draw a batch of data points and feed them to the model, we then get the out put of predictions, and finally we compare the predictions with the labels to get the accuracy. The work of backpropagation is taken care of by tensorflow. The parameters that we can tune are:

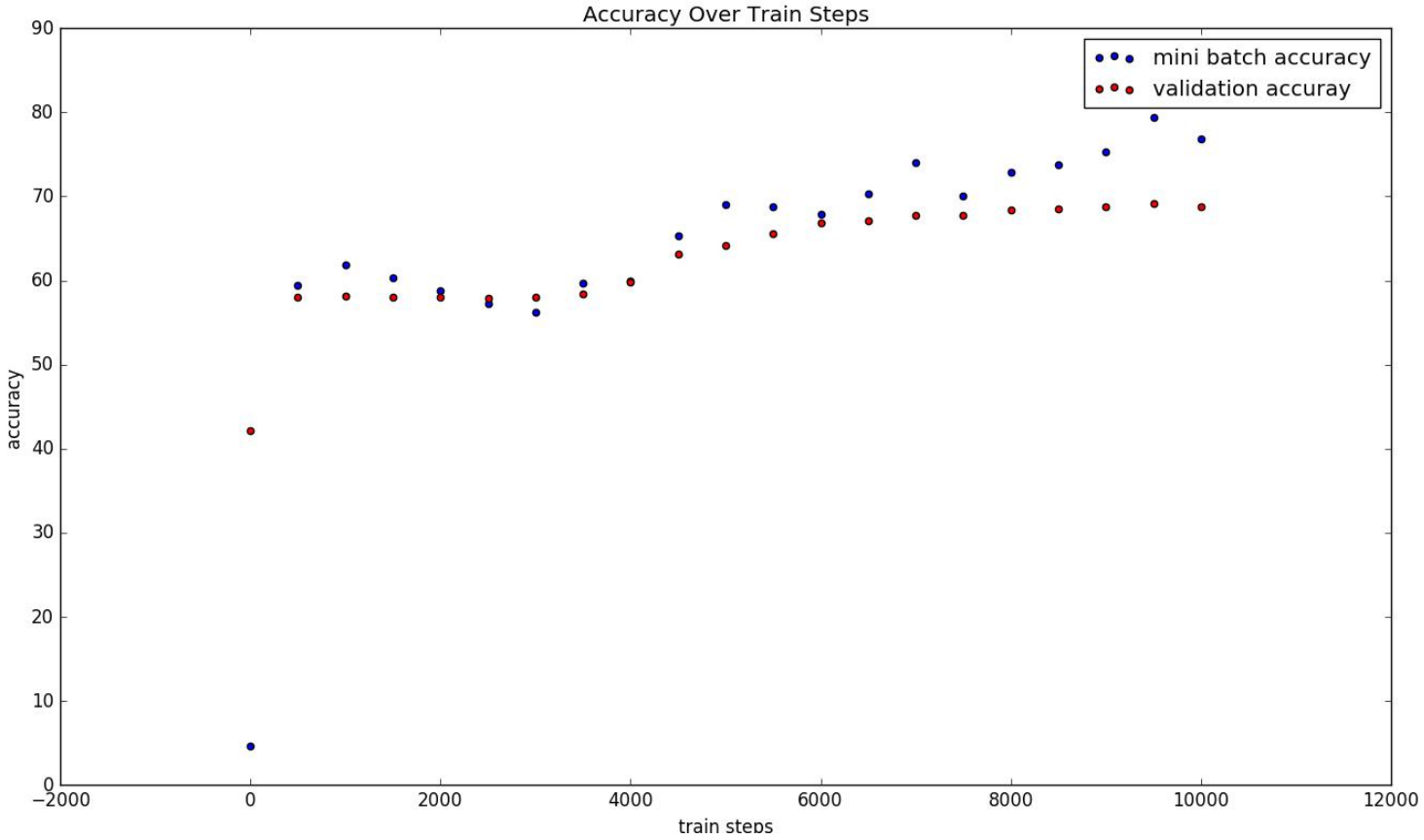
- Number of epochs
- Data points batch_size
- Learning rate
- Number of network layers
- Layer types
- Weights and biases of every layer

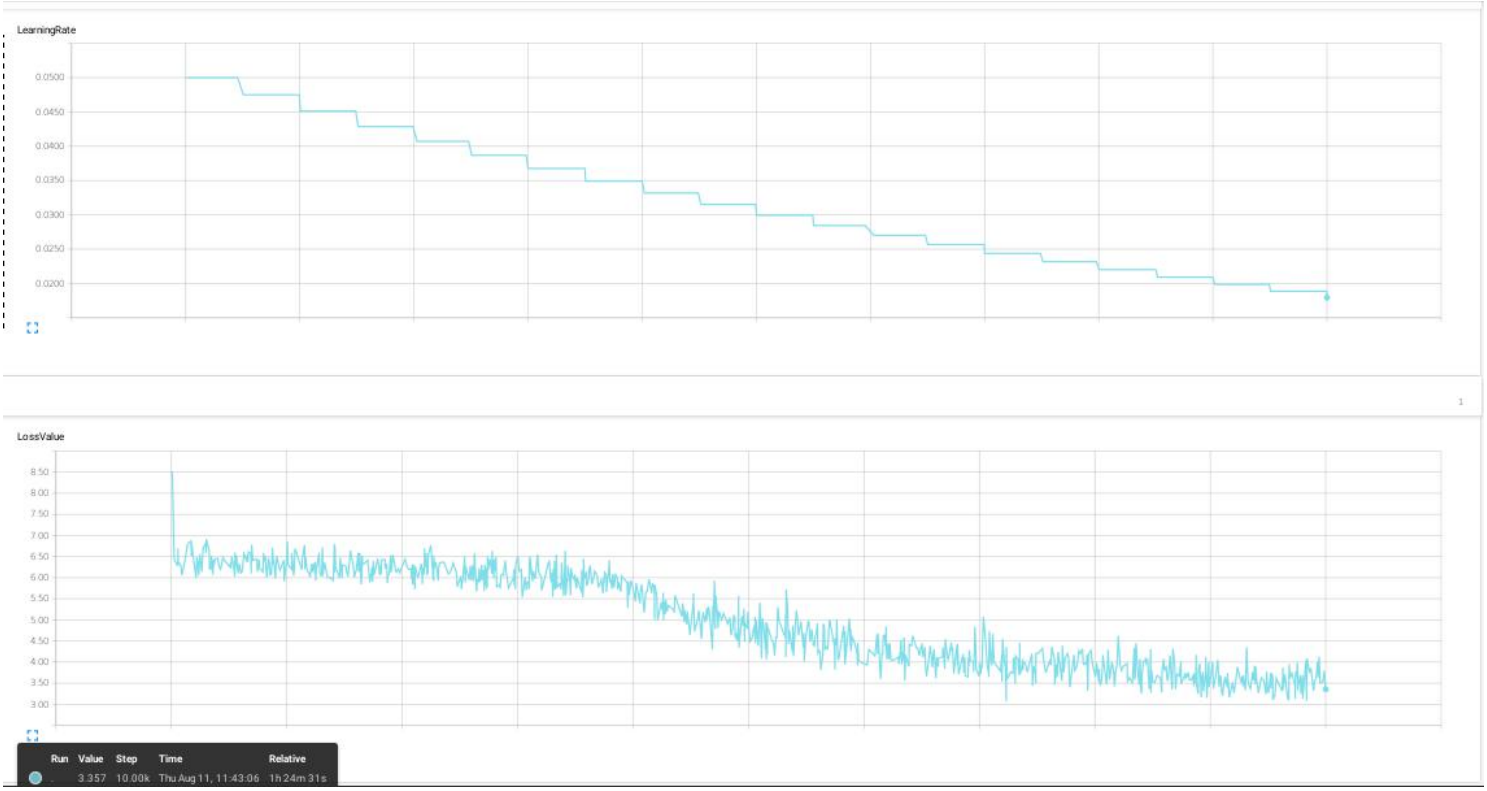
Benchmark

Here, the first figure below shows the model trained with 10 data points in 800 time steps, and it can overfit pretty quickly. As we can see from the plot the mini batch accuracy achieved 100% at around the 200th step and kept it that way afterward, but the validation accuracy is at around 40% and has not changed much, hence the overfitting. The second figure below shows the model trained with the whole dataset and 10001 step. This means the

model can learn, but the accuracy is not so good. The test accuracy is 72.7%, we will use this accuracy as a benchmark and the future performances will be measured against it. The third figure below shows the learning rate and loss over training time steps.




Accuracy Over Train Steps
mini batch accuracy
validation accuray
accuracy
train steps
90
80
70
60
50
40
30
20
10
0
−2000
0
2000
4000
6000
8000
10000
12000


LearningRate
0.0500
0.0450
0.0400
0.0350
0.0300
0.0250
0.0200
LossValue
8.50
8.00
7.50
7.00
6.50
6.00
5.50
5.00
4.50
4.00
3.50
3.00
Run Value Step Time Relative
3.357 10.00k Thu Aug 11, 11:43:06 1h 24m 31s

Methodology

Data Preprocessing

Even though the raw data from the SVHN dataset are all colorful pictures, in terms of current task, colorful pictures are redundant, gray scale pictures are just enough for this mission. So the data preprocessing consists of the following steps:

1. Get the information of image file name and respective labels and blue boxes from 'digitStruct.mat' files.
2. Use the blue boxes' information calculate and crop an area from the original image file, then convert it to gray scale image[3][iii].
3. After randomly shuffle the train and test dataset, store all the processed images in a whole pickle file.

Based on the Data Exploration section, we found there is only 1 data point has more than 5 digits, here we removed it from the train dataset before step 3.

Implementation

1. Structure of the Convolutional Neural Network.

Total network has 8 layers including input and output layers:

input --> conv_1 --> relu_1 --> conv_2 --> relu_2 --> pooling --> fully connected

--> output (logits).

The weights' shape of conv_1 is:

img_width X img_height X n_channels X depth1

The weights' shape of conv_2 is:

img_width X img_height X depth1 X depth2

The full connected layer has 128 nodes and the shape of the weights in this layer is 128 X 11.

2. Calculation of the accuracy.

The output of the model predictions' shape is:

n_digits X batch_size X n_labels

In order to compare this tensor with the respective labels, we first use numpy.argmax to find the predictions with the highest probability, then transpose the tensor so that it has the same shape with the respective labels, and finally calculate the accuracy.

3. The way to deal with the digitStruct.mat file.

These files are the key for the data processing. We use a class to aggregate all the logic in a place, in order to do this we need a python package 'h5py'. After initialized an instance of this class, we call the getBoxesAndLabels() and getFileName() methods to get the information we need to genrate the train and test data.

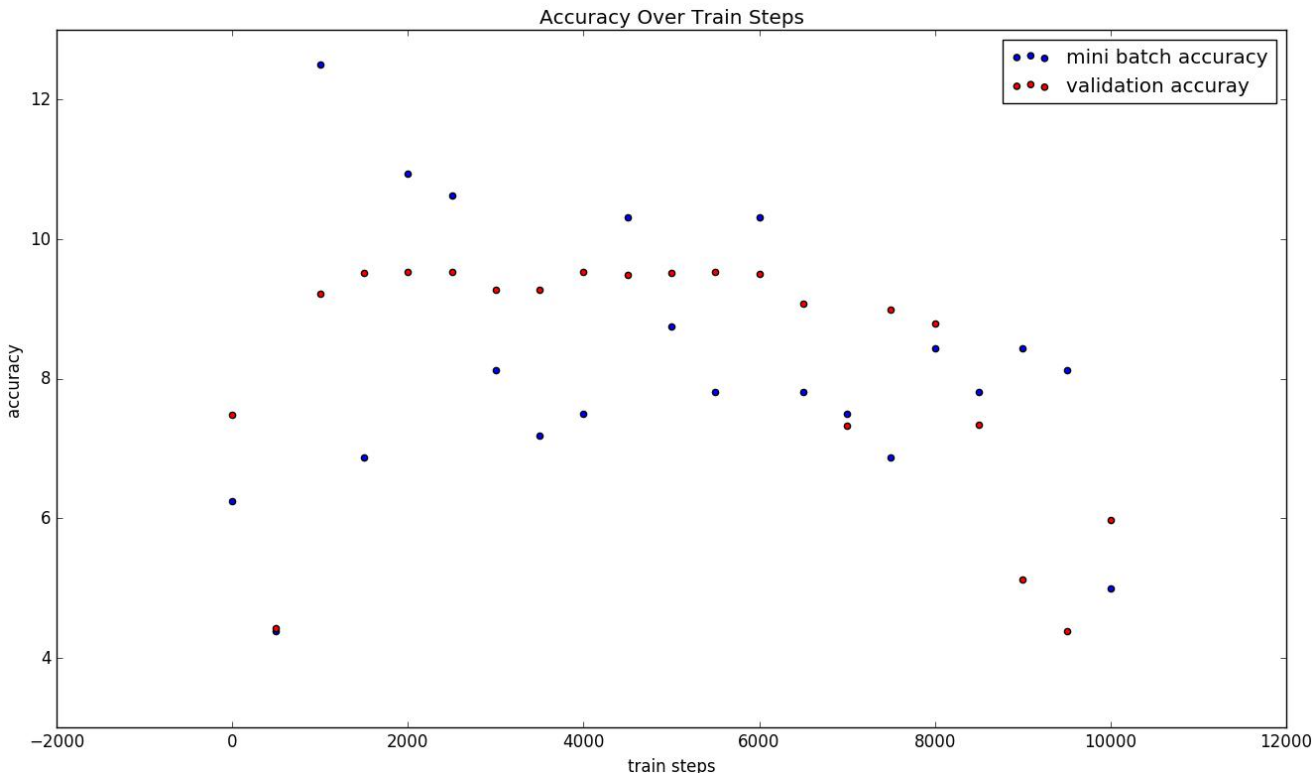
4. Toy dataset.

Of course, this part depends. My computer has only 4g memory, and the whole computation of the data flow through the network is too heavy for it

to handle, so I made a toy dataset from the full dataset by randomly sampling from it. This way I can easily try all kinds of network structures and layers and tune the hyperparameters in a breeze. We draw randomly 10,000 data points from training dataset; 800 from validation dataset and 1,000 data points from testing dataset, and use these three group respectively as toy_train_dataset, toy_valid_dataset and toy_test_dataset. The mini-dataset helps a lot at the beginning when I try to implement the network structure. The layers in the network may not couple with their neighbors, so I have to tune the shape of the weights to make sure the network structure works. With this mini-dataset, you can focus on what you really want to implement and easily make the trials. If you use the full dataset, it would be messy or even worse, impossible.

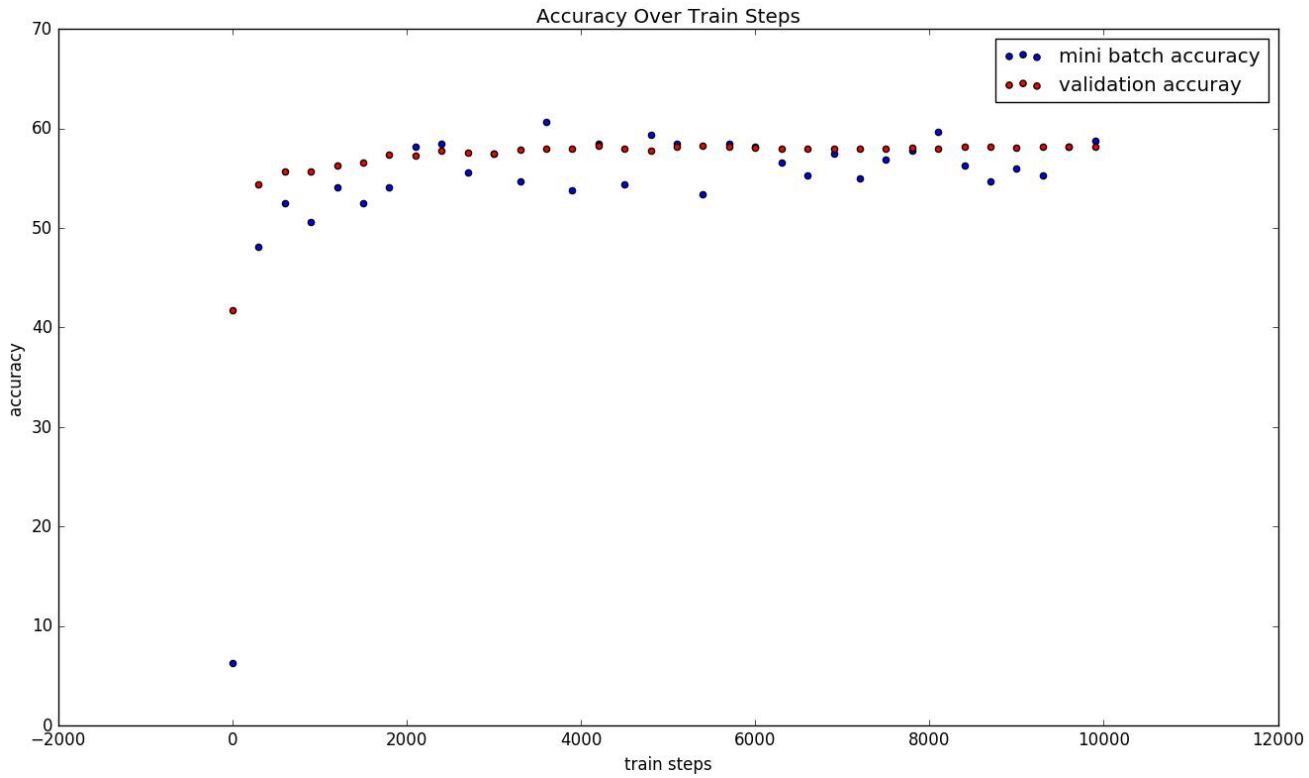
Refinement

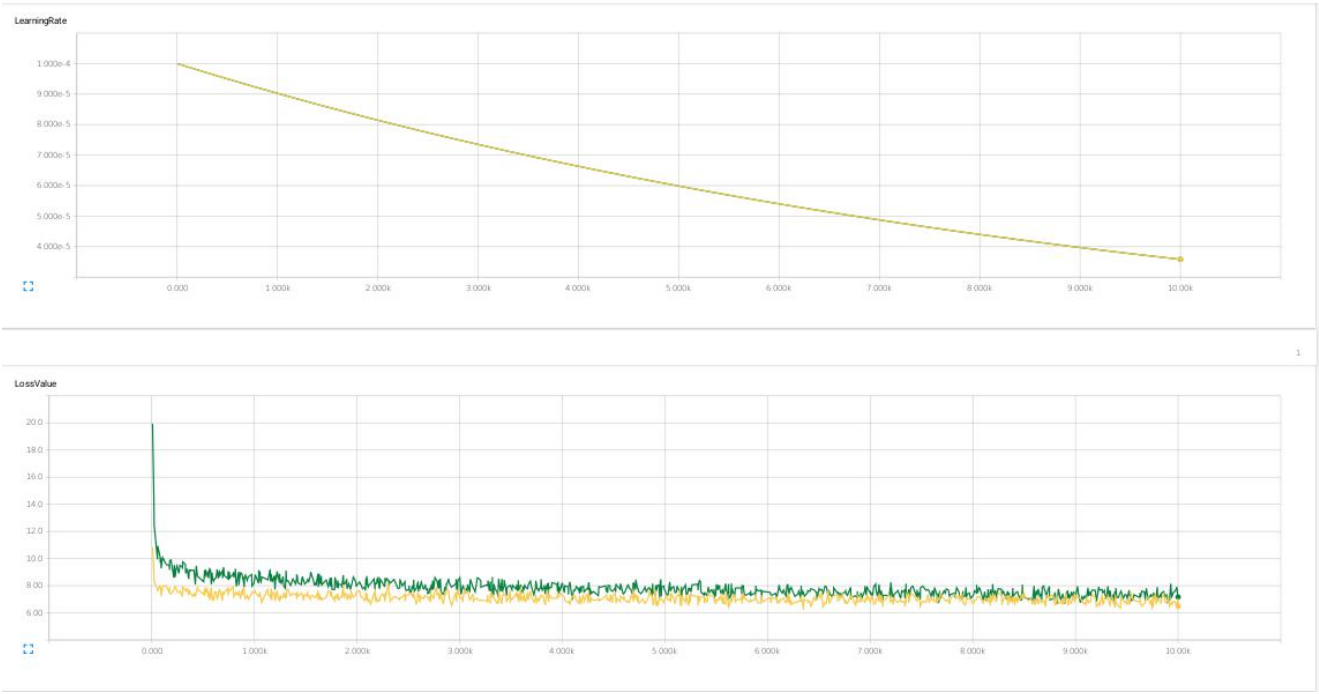
1. Initial solution:



The above figure shows the initial solution, the accuracy may be less than randomly guess. It seems the model may be doing nothing at the first sight, it surely is, however, a clear sign of that the code is functioning. From it, we know that all the layers in the model could couple well with their neighbors, and what we need to do is try to adjust the network architecture and tune the hyperparameters to get higher accuracy.

2. Intermediate solution





The above two figures show one of the intermediate solutions. The one after this is chosen as the benchmark of the model performance. The model structure, which produced the figures above, is changed to:

input --> conv_1 --> relu_1 --> pooling_1 -->

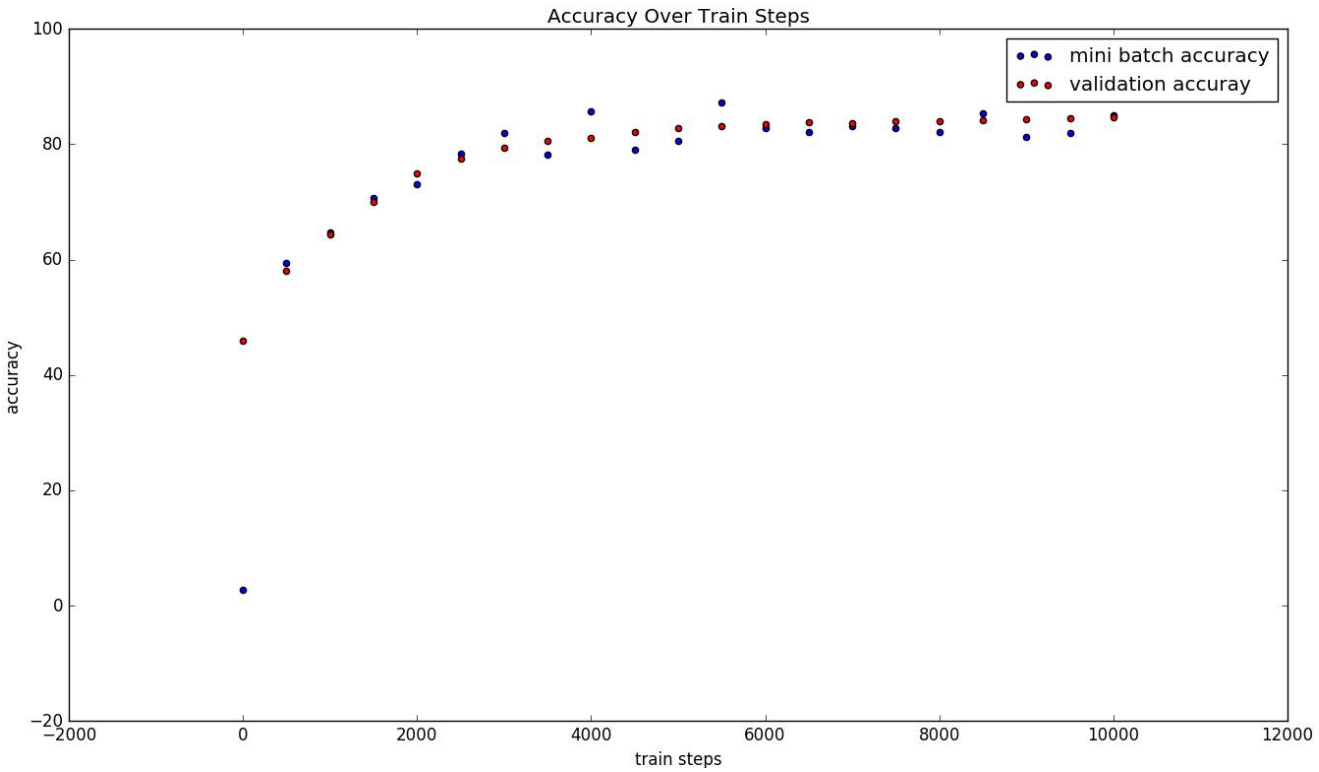
conv_2 --> relu_2 --> pooling_2 -->

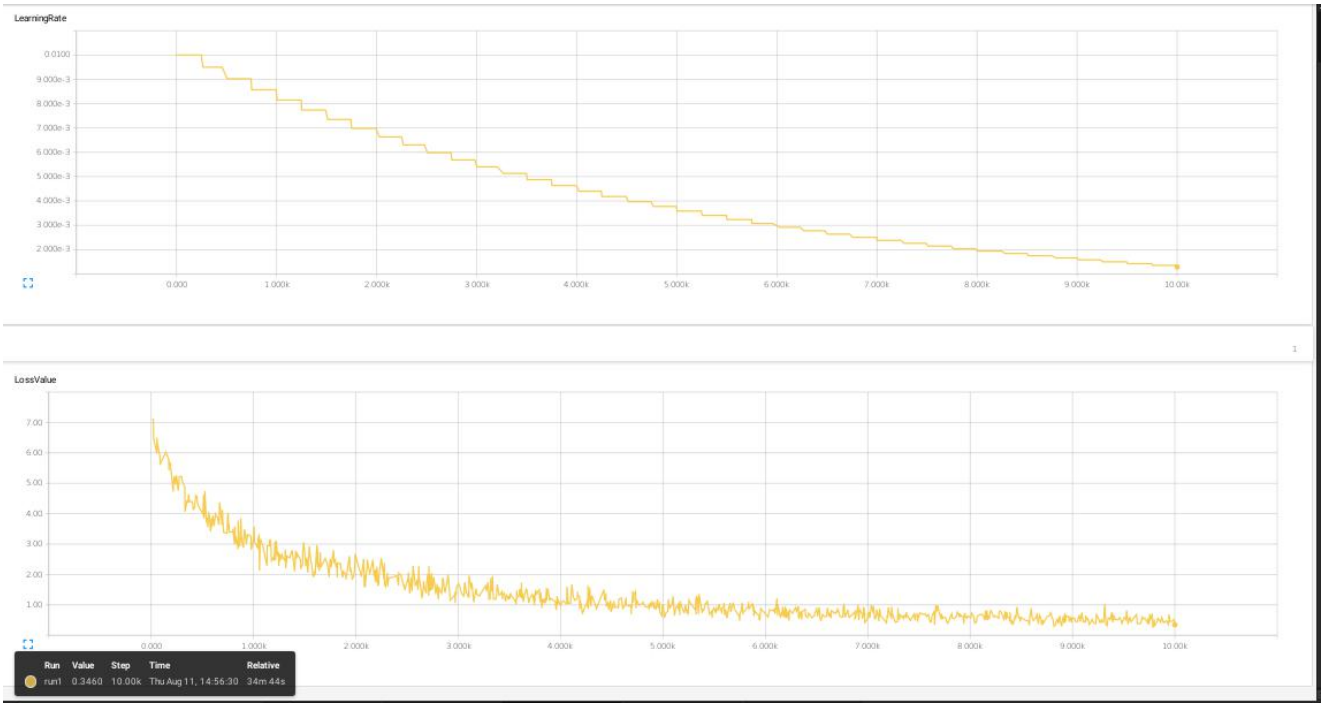
conv_3 --> relu_3 --> pooling_3 -->

fully connected ----------> output (logits)

The number of nodes in fully connected layer reduced from 128 to 64 and the learning rate is exponential decaying which starts at 1e-4. And changed gradient descent optimalization algorithm from SGD to Adagrad.

3. Final solution





The model produced the two figures above has the same structure as the intermediate solution's, only the decaying learning rate starts at 1e-2

Results

Model Evaluation and Validation

The model's evolution process is as follows:

n_layers	n_FC_nodes	fileter_size	Learning_rate	test_accuracy
8	128	3	1e-3	62.6%
12	64	5	5e-2	72.7%
12	64	3	1e-3	65.0%
12	64	3	1e-4	70.1%
12	64	3	1e-2	80.6%

The orange one was chosen as the benchmark: test accuracy 72.7%, the last green line was the final solution. We can see the final test accuracy was

improve by 7.9% on the base of the benchmark. I consider this is a significant improvement.

Model robustness:

The pictures from SVHN are the original, variable-resolution, color house-number images with character level bounding boxes, as shown in the examples images. 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 images). The images from this dataset have shown:

1) data quantity in abundance for the project

2) they are from natural scenes, hence the sufficient variance

While it's always good to do 'data augmentation', it is, I think, not necessary to do so under this circumstance. The model has shown enough robustness by the fact that it can handle the natural scene images.

Justification

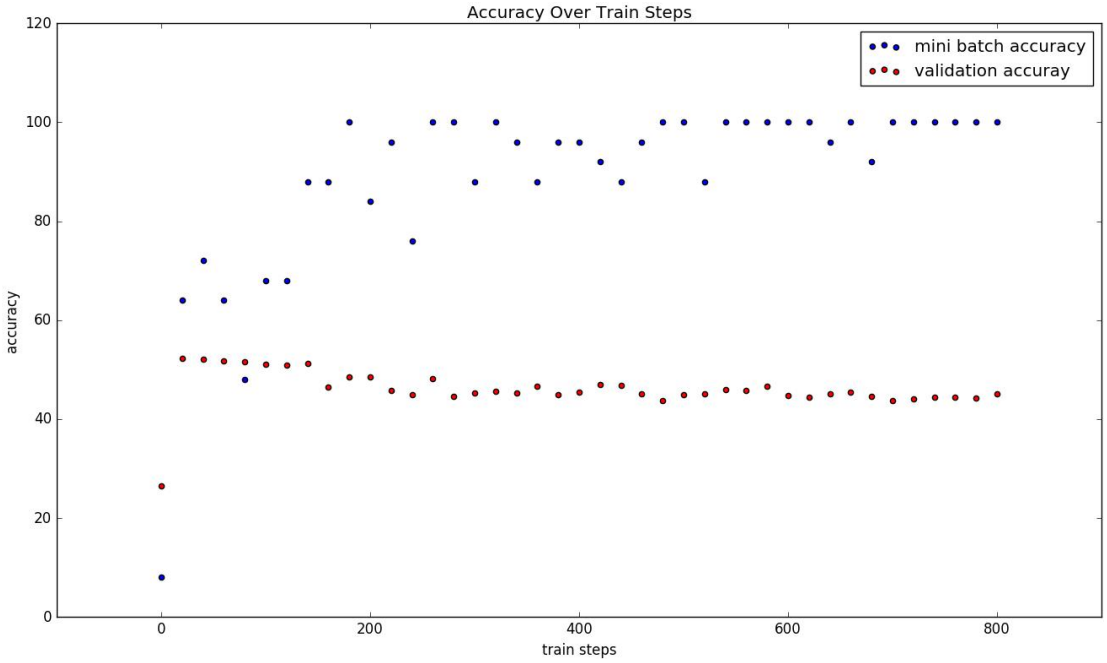
We can see from the figures and table in the last section, the final solution is much better than the benchmark. My test labels formed a 5000 X 5 tensor, this means there are 25000 digits there, improved by 7.9% means the final solution correctly recognized more 1975 digits than the benchmark. When compared with human performance, of cause, it is still not good enough. However, when we observe the scatter plot of final solution more closely and carefully, we find the model shows no sign of overfitting. This

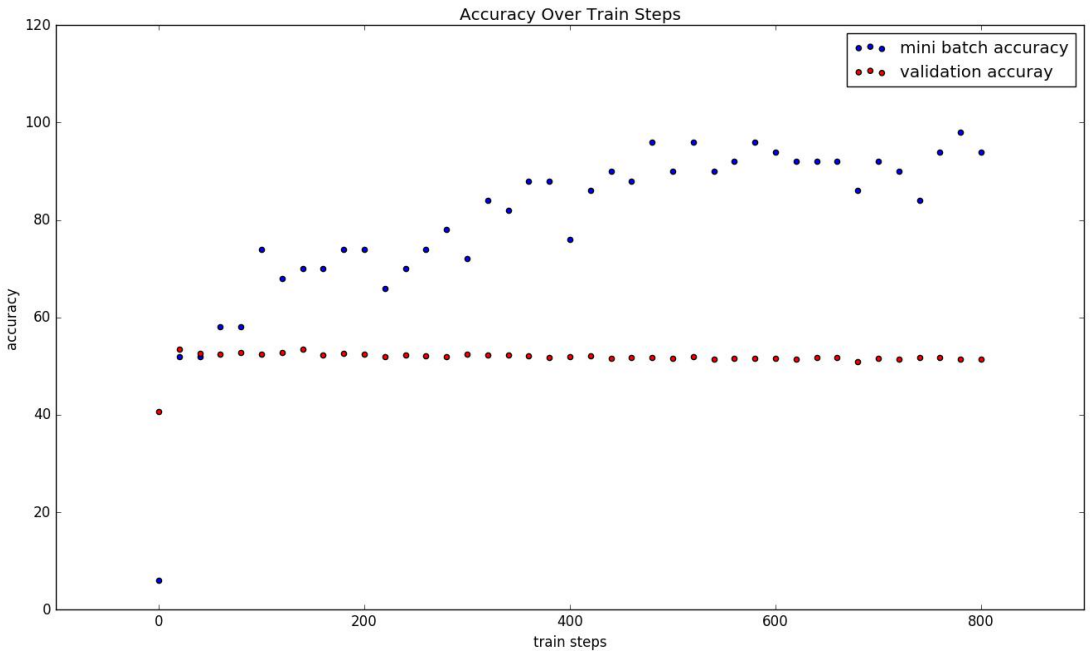
implies it can still learn more to make progress, hence the higher future accuracy.

Conclusion

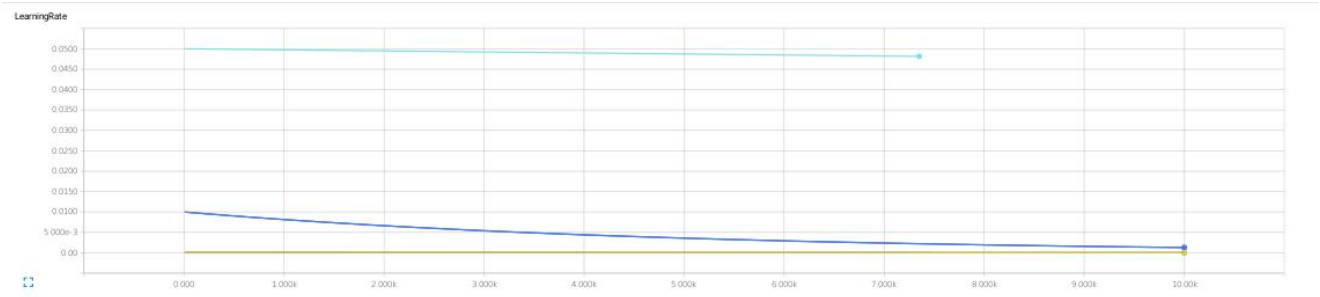
Free-Form Visualization

Here are some of the figures I collected during my exploration of the model structure and hyperparameter tuning.





These two figures above shows the model can learn, not in a very efficient stable way, but still. When I feed the initial implemented network structure with batch_size=32, it produced these 2 figures.



The last figure above tells us with different learning rate and the respective cross entropy loss over train steps. This means the learning rate is vital to the model's performance.

Reflection

The whole process of this project comprise these steps:

1. Data preprocessing.

This is the very foundation of the whole project. If you can't process the data for the latter steps, you simply can't start the project. Or worse, you did it wrongly. So you started it, but like we all know, garbage in garbage out. You just wasted your time and got the wrong conclusion.

2. Model implementation.

What is the appropriate network structure for my intention? I think this is the most difficult problem of all. I just can't answer it. Maybe there is only this heuristic rule: the bigger the better, or the deeper the better?

3. Model training and hyperparameter tuning.

Here comes all the bells and whistles of machine learning. And I think this is the most interesting part of the whole project. You ring this bell, the model gives you something, you blow that whistle, it gives another thing. Howerver, it is worth noting that since Convolutional Neural Networks are not scale invariant, after I used the toy dataset to confirm the structure of the model works, I switched to the full dataset. Then I pay attention to the model performance, which was impacted by this. And at last I managed to get the final solution.

Improvement

First and the most obvious way is to simply run more steps, because it shows

no sign of overfitting.

The image size. During my training process, the original image size is 32 pixels. However, now we think about it, the size of 28 pixels is just enough. This way we can reduce considerable amount of computational resources.

Adadelta the algorithm. I have not learned this algorithm yet, but I'm definitely interested. After I learned it in the coming days, I will use it. My first implementation used Stochastic Gradient Descent, then I switched to Adagrad and get a better performance. I think Adadelta may give me higher accuracy here.

Training droupout. If I could get larger and deeper network structure, I would definitely use it. But this needs more computational resources.

[i] [1] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, Vinay Shet. Multi-digit Number Recognition from Street View Imagery Using Deep Convolutional Neural Networks.

[ii] [2] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, Andrew Y. Ng Reading Digits in Natural Images with Unsupervised Feature Learning NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011.

[iii] [3] Mark Grundland, Neil A. Dodgson. Decolorize: Fast, Contrast Enhancing,Color to Grayscale Conversion.