Machine Learning Engineer Nanodegree

Capstone Project

Dafeng Jin
Augest 30st, 2017

I. Definition

Project Overview

Recognizing multi-digit numbers in photographs captured at street level is an important compo-nent of modern-day map making. A classic example of a corpus of such street level photographsis Google's Street View imagery comprised of hundreds of millions of geo-located 360 degreepanoramic images. The ability to automatically transcribe an address number from a geo-locatedpatch of pixels and associate the transcribed number with a known street address helps pinpoint,with a high degree of accuracy, the location of the building it represents.

More broadly, recognizing numbers in photographs is a problem of interest to the optical charac-ter recognition community. While OCR on constrained domains like document processing is wellstudied, arbitrary multi-character text recognition in photographs is still highly challenging. Thisdifficulty arises due to the wide variability in the visual appearance of text in the wild on accountof a large range of fonts, colors, styles, orientations, and character arrangements. The recognitionproblem is further complicated by environmental factors such as lighting, shadows, specularities,and occlusions as well as by image acquisition factors such as resolution, motion, and focus blurs.

In this project, I will train a deep learning model to automatically recognize digits within images from real-world images taken from a webcam. The application uses a Deep Convolutional Network trained using the SVHN dataset. This project is a capstone project in the Deep Learning course.

Problem Statement

In this project, I will ignore the MNIST dataset and use the realistic dataset - SVHN, with the following steps.

- Data preparing

  In this step, I will download the SVHN dataset and extract the data. After that I will parse the mat file and record the data

- Data preprocess

  In this step, I will do 3 things to preprocess the dataset.

  1、I will get the bounding box surrounding all digits and expand it by 30% the crop it. After that resize to cropped image to 32x32.

2、I will shuffle the data and split into three parts: Train, Valid and Test. Because the train dataset is a little large, I will use binary files to save it.

3、I will make the images gray scale and apply random brightness/contrast transformations to speed up the training process.

- Model

  In this step, I will train a convolution neural networks.

- Evaluation

  In this step, I will evaluate the model using the test dataset to see how well it performs.

Metrics

I will determine the accuracy of my Model by computing the proportion of the testing images for which the length of the sequence and every element or digit in the sequence is predicted correctly. There will be no partial credit for correctly classifying individual digits.

II. Analysis

Data Exploration

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.

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.) Each tar.gz file contains the orignal images in png format, together with a digitStruct.mat file, which can be loaded using Matlab. The digitStruct.mat file contains a struct called **digitStruct** with the same length as the number of original images. Each element in digitStruct has the following fields: **name** which is a string containing the filename of the corresponding image. **bbox** which is a struct array that contains the position, size and label of each digit bounding box in the image. Eg: `digitStruct(300).bbox(2).height` gives height of the 2nd digit bounding box in the 300th image.

Exploratory Visualization

- Data size

The SVHN dataset is composed not only by training and testing samples but also by an additional, somewhat less difficult, extra set which can be used as additional training. The number of images which are available on each dataset is shown below.

Dataset	**Size**
Train	33402
Test	13068
Extra	202353

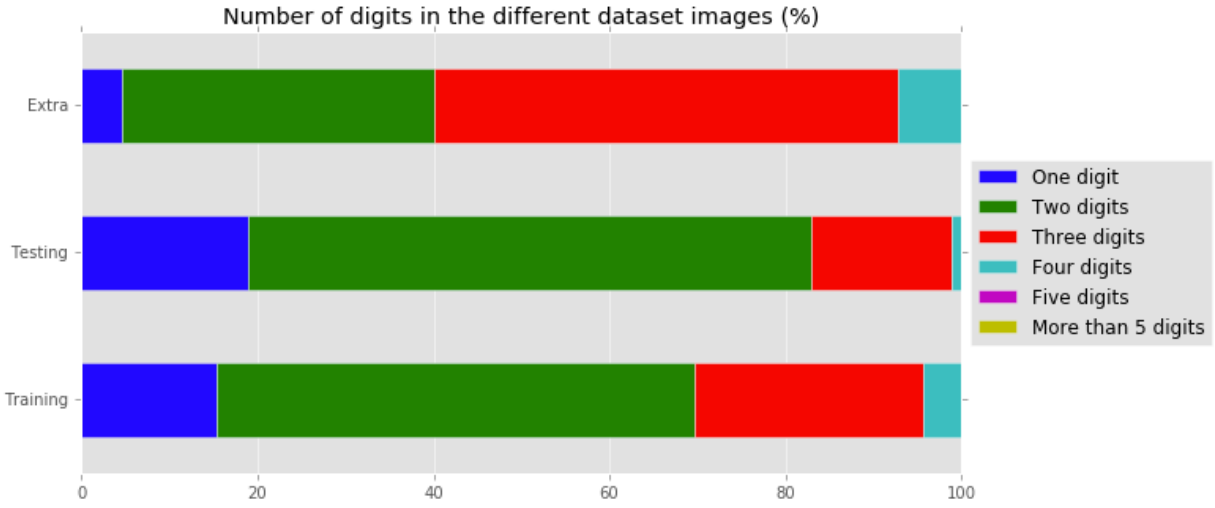
- Statistics

Some statistics regarding the size of the images.

	Avg. Height	**Avg. Width**	**Max. Height**	**Max. Width**	**Min. Height**	**Min. Width**
Training	57.213011	128.284983	501	876	12	25
Testing	71.566498	172.583486	516	1083	13	31
Extra	60.800151	100.389250	415	668	13	22

- Number of digits

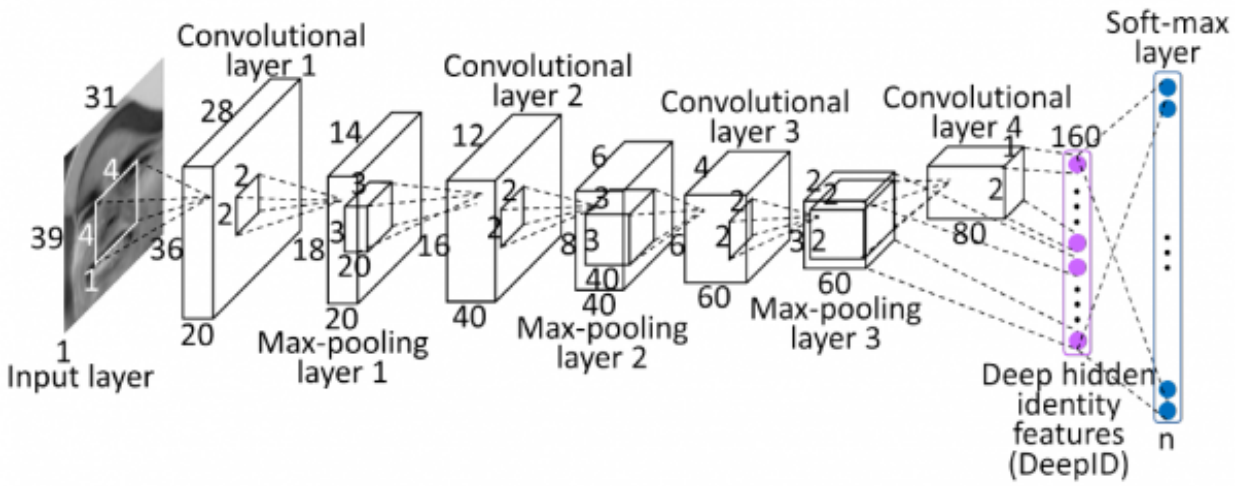
It is also interesting to analyze the length of the number strings which can be found in the available images. Knowing the distribution of number string lengths can help us to determine if the dataset is useful for us or not. Below shows the percentage of the datasets which contains number strings with a given length.



The datasets are composed mainly by number strings with up to 4 digits. Based on these results I will design the network to deal with up to 5 digits. Given the lack of training examples, however, I do not expect the network to perform well on images containing 5 numbers.

Algorithms and Techniques

I'll be using a Convolution Neural Network to solve the problem. CNNs can represent very complex models and are currently the state-of-the-art technique for most image processing tasks. CNNs are very similar to regular neural networks but their architecture is designed to take advantage of the 2D structure of the input image (or a 2D input such as audio signals). A typical CNN architecture consists of several convolutional and subsampling layers followed by fully connected layers.



- Images / Input Layer

This layer holds the the input which is equivalent to a store of the the pixels of all images.

- Convnet / Convolution Layer

Convnets are neural networks that share their parameters across space. It works by sliding over the vector with depth, height and width and producing another matrix(called a convolution) that has a different weight, depth height. Essentially, rather than having multiple matrix multipliers, we have a set of convolutions. For example, if we have an image of size 1024x1024px in 3 channels, we could progressively create convolutions till our final convolution has a size of 32x32px and a much larger depth.

- Pooling (Max Pooling)

Pooling is a technique that can be used to reduce the spatial extent of a convnet. Pooling finds a way to combine all information from a previous convolution into new convolution. For our example, we'll be using maxpooling, takes the maximum of all the responses in a given neighborhood. We choose it because it does not add to our feature space and yet gives accurate responses.

- Fully Connected Layer

This layer connects every neuron from the previous convolutions to every neuron that it has. It converts a spatiallike network to a 1d network, so we can then use this network to produce our outputs.

- The Output layer

The output from this layer are logits which represents matrix showing the probabilities that of having a character in a particular position.

Benchmark

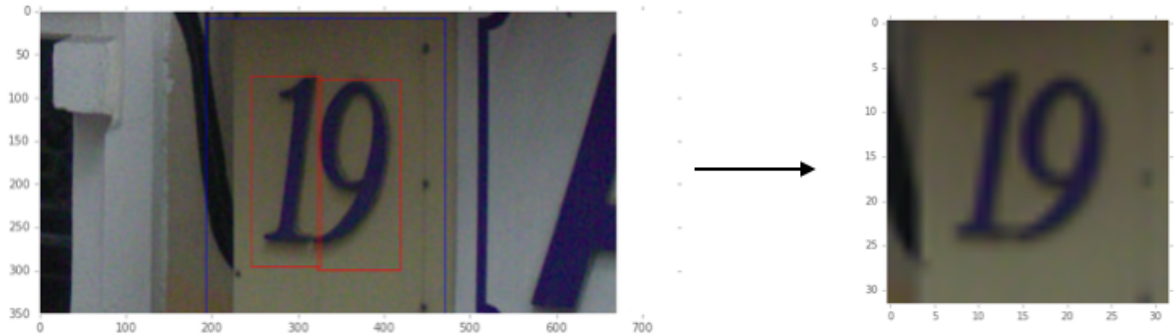
Because of lack of computing power, I will implement some more recent techniques to reduce the model complexity without affecting its performance too much. I will focus on optimizing the network architecture. My ideal performance would be obtaining a transcription accuracy about 85%.

III. Methodology

Data Preprocessing

I will do 3 things to preprocess the dataset.

1、I will get the bounding box surrounding all digits and expand it by 30% the crop it. After that resize to cropped image to 32x32.



2、I will shuffle the data and split into three parts: Train, Valid and Test. Because the train dataset is a little large, I will use binary files to save it.

3、I will make the images gray scale and apply random brightness/contrast transformations to speed up the training process.

Implementation

All the code was implemented as Jupyter Notebooks. For simplity I just split the project into two notebooks.

- 1_Data_Preparing.ipynb

This will do most work of data preprocessing

- 2_Model.ipynb

This will do the model architecture, training and evaluating. I will use the model as below:

- C 1: convolutional layer, batch_size x 28 x 28 x 16, convolution size: 5 x 5 x 1 x 16
- S2: subsampling layer, batch_size x 14 x 14 x 16
- C3: convolutional layer, batch_size x 10 x 10 x 32, convolution size: 5 x 5 x 16 x 32
- S4: subsampling layer, batch_size x 5 x 5 x 32
- C5: convolutional layer, batch_size x 1 x 1 x 64, convolution size: 5 x 5 x 32 x 64
- Dropout F6: fullyconnected layer, weight size: 64 x 16
- Output layer, weight size: 16 x 10

I read the preprocessed data into the model and train it in batches. During the training, we try to minimize loss and log the accuracy we are achieving so we can keep track of how well our model is improving. Once the model is trained, we evaluate it using our test step

Refinement

The following parameters were also modified in an iterative way trying to optimize the performance on the validation set while avoiding overfitting:

- Number of convolutional layers: I tried models with 4 and 5 convolutional layers but they make the training process too slower for my computer. So I finally choose 3.
- Depth of the convolutional layers and nodes in the fully connected layer: I increased the number of channels on each layer trying to find a good threshold between performance and training time.
- Learning rate: I had some problems when starting the project and it was all due to a large learning rate. The model was overshooting during training and the loss was not decreasing even after training for many steps. It took me some time to find out the reason, but once I did I could find optimal values.
- Dropout: The dropout probability was also tuned after trying a couple of different values.

IV. Results

Model Evaluation and Validation

The benchmark data are:

- Minibatch loss: 2.060841
- Minibatch accuracy: 88.4%
- Validation accuracy: 85.8%
- Test accuracy: 86.4%

Justification

Human recognition is 97% while that of a naive classifier was below 60%. Our model was able to achieve 86.4% accuracy while training on 10000. This is much better than the naive classifier. Since we have over 200,000 more images to train on, we expect the accuracy to continue to improve moving closer to human recognition.

V. Conclusion

Free-Form Visualization

In the data preprocess, I make the images gray scale to make the data little and speed up the train process. But it bring some problems like below:

Even human being can't decide what the digits are. 1 or 11? So it must be careful when making changes to the dataset. Sometimes I think its more meaning than the traning model.

Reflection

In the process of the whole project, I find it mainly include two parts.

First part is the data preprocess, it include how we want the model to see the data. Sometimes data preprocess is so important that better present of data will easily improve the model and speed up the train process.

Second part is about tuning the model. Thanks to Tensorflow, we don't need to implement the algorithm. All we need to do is to understand the architecure and adjust the parameters. But I also see I have to buy a better computer to make the tunning process faster.

There is also another part about how to make the training process faster. It including optimize the directive of CPU and GPU. And make it run on distributed system.

I think machine learning really chain all aeras of computer science and I'm really happy to finish this project and look forward to meeting the coming AI challenges.

Improvement

I just finish the basic part of the project. There are many things to improve in this project.

- I don't use the MNIST for synthetic data. I think combine the MNIST and SVHN dataset may

make the data more abundant and improve the result.

- I don't make the process of localizing where the numbers are on the image. I want to try later.
- I don't deploy it to an Android app, I will do it once I'm familiar with Android development.