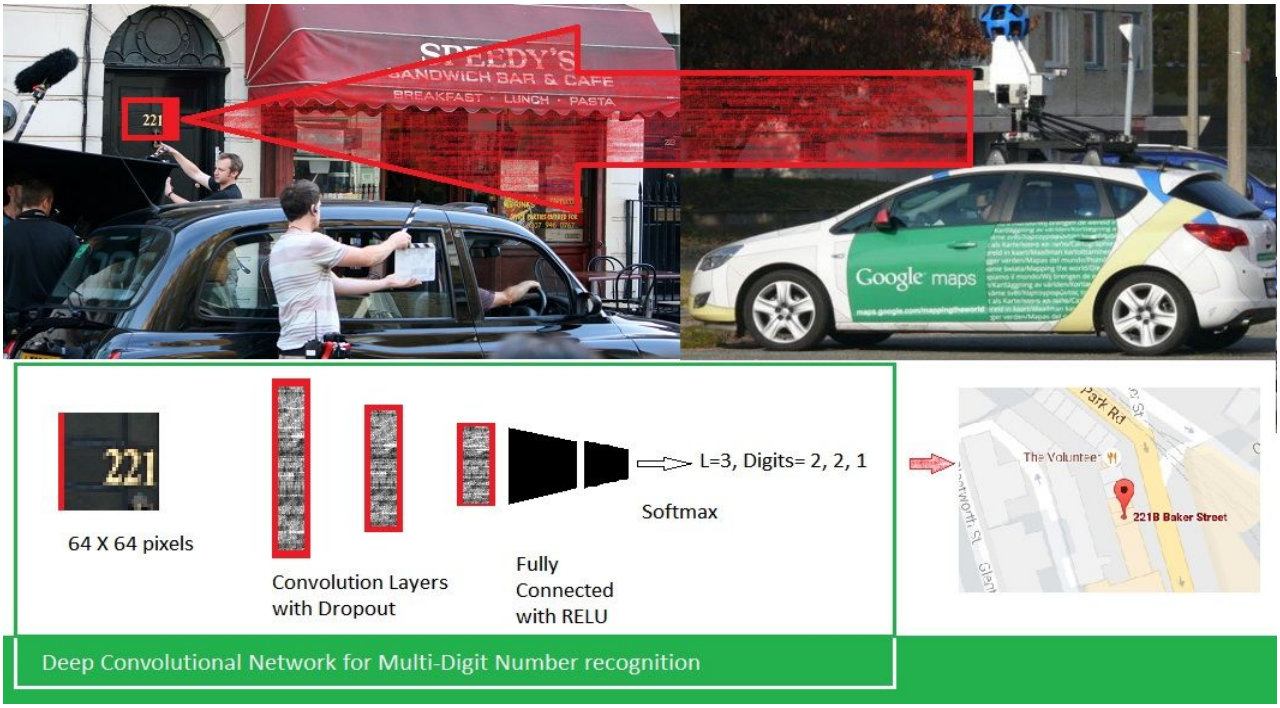
Capstone Project
Udacity Machine Learning NanoDegree

Deep Learning using TensorFlow to recognize house numbers in Street View House Number Images

- Implementation of Google's algo on multi-digit number recognition using Deep CNN

Sample use-case of multi-digit number recognition to put house numbers on Map



Definition

Project Overview

Most people and machines capture images of our surroundings on a daily basis. Many of these images contain numbers as a sequences of digits, in a natural and unconstrained environment. These digits often do not conform to standard fonts or typography.

This project provides a basic implementation to automate recognition of such sequence of digits from natural images.

Problem Statement

Data: In our endeavour to identify multiple digits in images of natural scenes, we will use Street view House Number data [REF: #1] for training of our model and also to test it.

Model: To learn of shapes of digits in unconstrained images, we will employ 8 Convolutional Neural Network layers as it allows spatial translation. And have 2 fully connected layer and six softmax classifiers to predict different digits and length of number sequence in each image.

Assumption here is that maximum number of digits in sequence will not be more than 5.

Performance Metrics

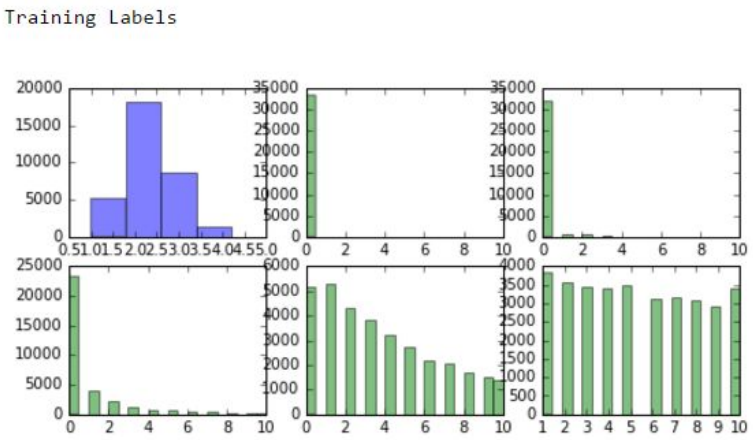
Accuracy: Effectiveness of model depends on how accurately it is able to predict the house number in test data. So performance of model will be based on accuracy in predicting length and digits of house numbers embedded in test image data.

Analysis

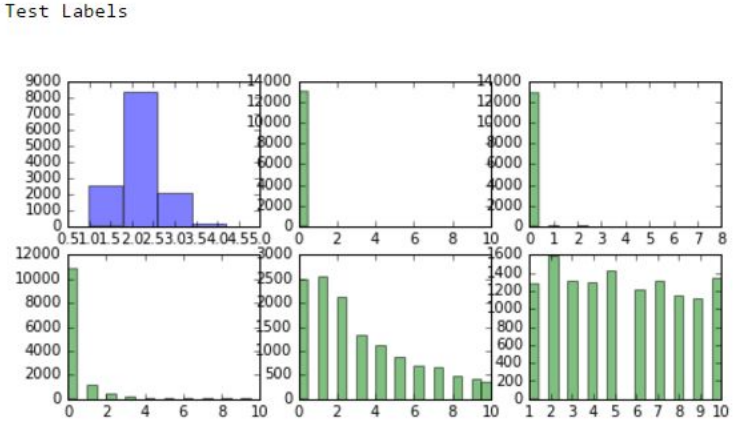
Data Exploration

Distribution of Digits and length of house numbers
House Numbers have different length, and have different digits at different places.
We want to compare the length of house numbers and distribution of digits in training images to that in test images.
There are 2 grids below. First grid is for training data and second grid is for test data.
Data points for training data and test data has been taken post normalization using pixel depth=255.
First plot in each histogram grid is of blue color and is a histogram of length.
Other plots in each grid show the distribution of each digits from 0-9 at each of the 5 places in this max-5-digit house number.

Training Data Histogram



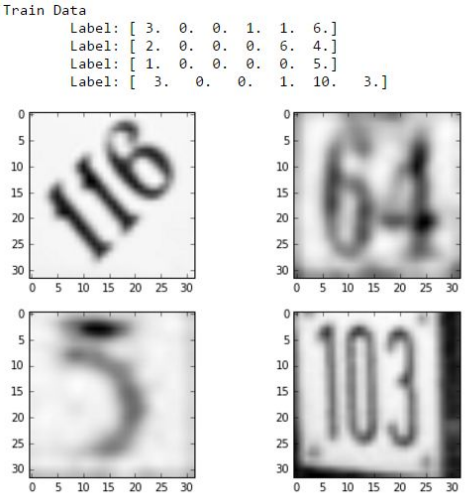
Test Data Histogram

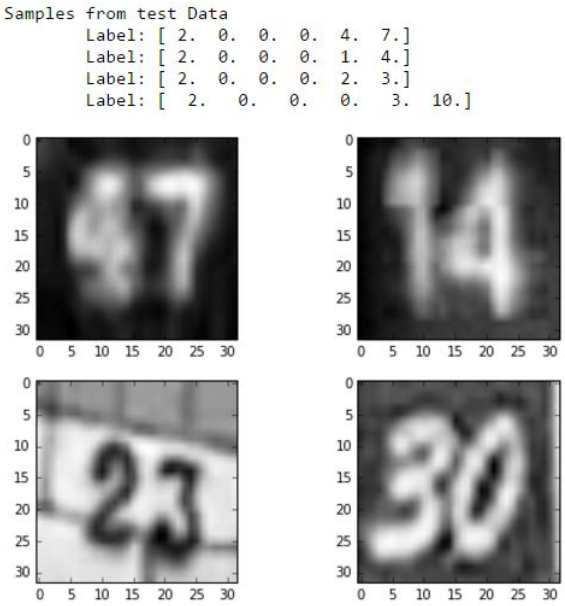


Similarity in above plots depicts that house numbers are in test data are similar to those in training data.

Exploratory Visualization

Here are visual samples for train and test data after normalizing original images and converting them to grayscale , that include labels and digits from house number.





Algorithm and Techniques

Following techniques will be used for this project:

a. Convolutional Neural Network layer - spatial translation to learn shapes.
b. Fully Connected Layer - to learn digits.
c. Dropout - to reduce overfitting.
d. Local Response Normalization - improves generalization, and reduces error rates.
e. Rectified Linear Units - allows learning non-linear curves.
f. Max Pooling - reduces complexity of model.
g. Gradient Descent Optimization - core classifier.

Benchmark

Proposed model will include prediction of length and individual digits of of house numbers. Due to limited computing resource, I intend to target for an accuracy of 75% on test data.

Methodology

Data preprocessing

SVHN data is available in 2 formats:

Format #1, is original variable-resolution image, along with with bounding box information.

Fotmat #2, is curated 32 X 32 pixels image, similar to MNIST data.

This implementation uses Format #1, to allow handling of more realistic image.

Pre-processing: SVHN Data in format #1 has 2 components: Images files, and Bounding Box information in matlab format. We use bounding box info in .mat file to localize whole sequence of number in an image, expand top, bottom, left, and right boundaries by 5%, and crop image to this dimension. Then we reshape it to 32 X 32 pixels, before converting to grayscale, and normalize it to zero mean, and unit standard deviation. This allows for better training.

Implementation

Following are the steps followed in implementation:

1. Data loading and transformation.
2. Pickling in required format.
3. Exploratory Analysis.
4. Reformat to required size, that is 32 X 32 X 1 format.
5. Define Model and train it.
    a. Define Accuracy function
    b. Define weights and biases.
    c. Create 8 convolutional layers and 2 fully connected layers.
6. Evaluate model on test data.

Problems faced and refinement done

1. Saved by RELUs: During early phase of model creation and training, I observed a pattern in predictions. All digits being predicted at an individual place, say all digits at place 3, were same. After much analysis, I was able to conclude this behaviour was emanating from inability of model to differentiate between different shapes. This is when RELU layers were added, to allow learning of non-linear shapes. After addition of RELUs, model was able to predict different digits at same place.

2. Skip missing digits: Goodfellow's paper mentions about skipping feedback from missing digits. Earlier attempt to implement this failed miserably, when I tried to mask output from FC layers while calculating logits. That attempt resulted in model unable to learn at all. After multiple readings of the paper and further analysis of code, I figured out that instead of logits, the loss being calculated in softmax layer had to be masked for missing digits. After this correction, model was able to improve further.

3. Use predicted length: Model being created included prediction of all 5 digits and length in calculating accuracy. Even after having 8 Convolutional layers and 2 fully connected layers, training accuracy was stuck at 82%, and test accuracy was stuck at 69%. Problem was that model was not using predicted length to consider only available digits for comparison. Instead the accuracy function was comparing prediction for all 5 digits, existing and missing, against the actual digits. This would not work since it would be hard to learn if a space at the beginning of house number image is for 1 missing digit or 2 missing digits; moreover these since length being used was predicted from model, so using that length to trim only those many digits is absolutely correct thing to do. Once this was done, training accuracy moved beyond 88%, and test accuracy reached 75%.

Refinement proposed

1. Train and learn more from examples that are have higher loss rate and lower accuracy during training..
2. Shuffle training examples after each epoch.
3. Add component to calculate bounding box.
4. Implement on multiple GPUs simultaneously.
5. Implement transfer of learning across multiple GPUs.

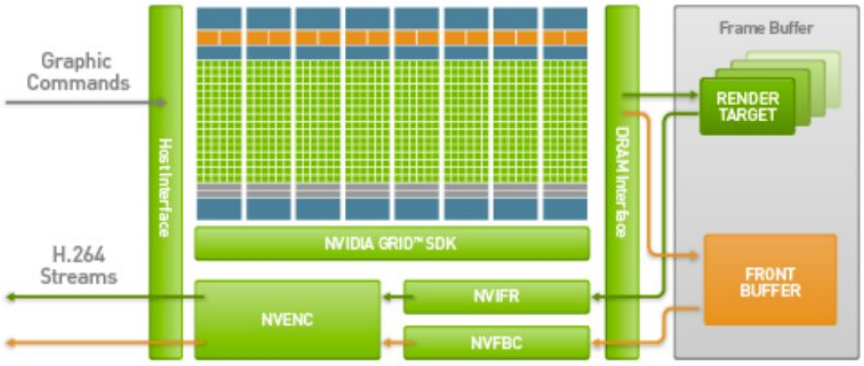
Results

Model Evaluation and Validation

Above described model was run on a AWS instance using GPUs. The new **g2.8xlarge**[REF # 6] instance has the following specifications:

- Four NVIDIA GRID GPUs, each with 1,536 CUDA cores and 4 GB of video memory and the ability to encode either four real-time HD video streams at 1080p or eight real-time HD video streams at 720P.
- 32 vCPUs.
- 60 GiB of memory.
- 240 GB (2 x 120) of SSD storage.

Here's a block diagram of the NVIDIA GRID GPU in the **g2** instance:



Pre-processing results:

Training Data:

image_labels: (33402, 6)
image_labels_array: (33402, 6)
Mean: -0.0566795406976
Standard deviation: 0.19967425856

Test Data:

image_labels: (13068, 6)
image_labels_array: (13068, 6)
Mean: -0.0510183666569
Standard deviation: 0.225976380396

Execution Results:

Here is the result of model execution:

```
Minibatch loss at step 25000: 0.001621
Minibatch accuracy: 100.0%
Minibatch loss at step 26000: 0.040764
Minibatch accuracy: 98.4%
Minibatch loss at step 27000: 0.003704
Minibatch accuracy: 100.0%
Minibatch loss at step 28000: 0.007553
Minibatch accuracy: 100.0%
Minibatch loss at step 29000: 0.003003
Minibatch accuracy: 100.0%
Minibatch loss at step 30000: 0.000352
Minibatch accuracy: 100.0%
Test accuracy: 78.9%
```

Conclusions

This project involved analyzing images of house numbers and extraction of exact house number from these images. There were several complexities associated including theoretical, tooling, and infrastructure challenges.

First the publicly available image data has to be structured in a form that could be operated upon by Python library, and in a efficient manner. Then this data had to be normalized for better results during Model training and Evaluation. Current implementation combines both these tasks in a single loop to reduce execution time. Only Train data set has been used, and Extra data set has been pickled but not used for modelling. Using Extra data, and generating more data by rotating images could improve accuracy further.

Creating an appropriate model posed several challenges. Although the paper suggests using 5 CNN layers, I experimented with variable number of layers to see the results and let the model be derived from requirements, instead of from theory. Starting with 2 CNN layers resulted in learning of missing digits, but the model predicted only similar numbers, like all 1s even when a digit is actually 7.

Real benefit came from adding RELU activations after CNN layers, which resulted in model being able to identify different shapes at different places in same house number. Adding Dropout helped reduce overfitting.

Current model uses 30,000 iterations, each having a batch size of 64. Kernel size was 5 for all layers.

There are 10 CNN layers and 2 FC layers. Here are their depths:

depth1 = 16
depth2 = 24
depth3 = 32
depth4 = 48
depth5 = 64
depth6 = 128
depth7 = 128
depth8 = 256
depth9 = 512
depth10 = 1024
num_hidden1 = 1024
num_hidden2 = 512

Executing such a large graph took 6 hours on AWS g2.8x large instance.

I intend to improve this further with additional synthetic data from same seed image data, by adding rotation and randomly trimmed images. Also plan to add plots of tensorflow execution summaries, for a better view into runtime.

References

1. Format 1 of Street View House Number (SVHN) from Stanford data: http://ufldl.stanford.edu/housenumbers/
2. Google's paper - Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks; http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42241.pdf
3. MNIST Data of Handwritten Digits: http://yann.lecun.com/exdb/mnist/
4. Dropout to prevent overfitting: http://www.jmlr.org/papers/volume15/srivastava14a.old/source/srivastava14a.pdf
5. AWS GPU INstances: https://aws.amazon.com/blogs/aws/new-g2-instance-type-with-4x-more-gpu-power/