Capstone Project

Machine Learning Engineer Nanodegree

Created by Baibhav Kumar Ojha

November 9, 2019

I. Definition

Project Overview

In this project, we will be training a deep learning model to automatically recognize digits within an image. The science here can be applied to a large scale problem, such as recognizing house numbers from google street view images. We noticed that a housing number is a sequence of digits. To help understand this, we can build a classifier that understands how to classify digits and use a convolution neural networks to scan through an image, returning only the parts that it believes contains a digit.

Problem Statement

In this project, we are going to train a machine learning model to decode a sequence of digits within a natural image. To do so, we'll first create a synthetic dataset from the MINST dataset and use that to train our model. We'll then validate the model synthetic test data. Once we are satisfied with the results, we'll validate that the model performs well using the SVHN dataset. If it performs well, we would have done a great job. If it doesn't, we'll review our model and learn what was not working.

STEP 0: Exploring using the 32x32 pixels images

The first step followed was to explore the dataset with simple fully connected neural networks. We would create a validation set from train and test by randomly selecting a few entries. We then use these values to train the network.

STEP 1: Synthetic Data Creation

We would create a synthetic dataset by concatenating digits in the MNIST dataset. These values would then be split into train, validation and test dataset.

STEP 2: Data Extraction.

The goal here is to extract data and make it meaningful for deep learning. We could use Gaussian normalization to ensure that our dataset gives us the best results. If the images are not the same size, we would have to resize the images to all have the same size.

Step 3: Modelling

Here, we'd create a convolution neural networks using a similar build up as Alex net. We would then train our network using the training and validation synthetic dataset.

Step 4: Evaluation

We'd then evaluate the model using the test dataset to see how well it performs.

Step 5: Testing with Live data

We'd test the network we have trained on live data from the SVHN dataset. If it does not perform well, we'd retrain the network using the SVHN dataset and reevaluate the network's performance

Step 6: Deployment

We'd deploy the model to either an android app or a web service.

Metrics

Since this problem is quite delicate - we don't want to mislead people with wrong house numbers, we would only give credit when all digits in the sequence are predicted. We would use the percentage of correctly predicted sequences to measure the performance of our model

II. Analysis

Data Exploration

The dataset has been separated into three. One for train, test and extra data. Also an additional dataset is provided which contains cropped 32x32 pixel images. With this additional dataset, we can run experiments such as train a logistic regression model and see how it performs. The original exists as a unzipped file which once extracted contains the raw images as well as a digitStruct.mat file.

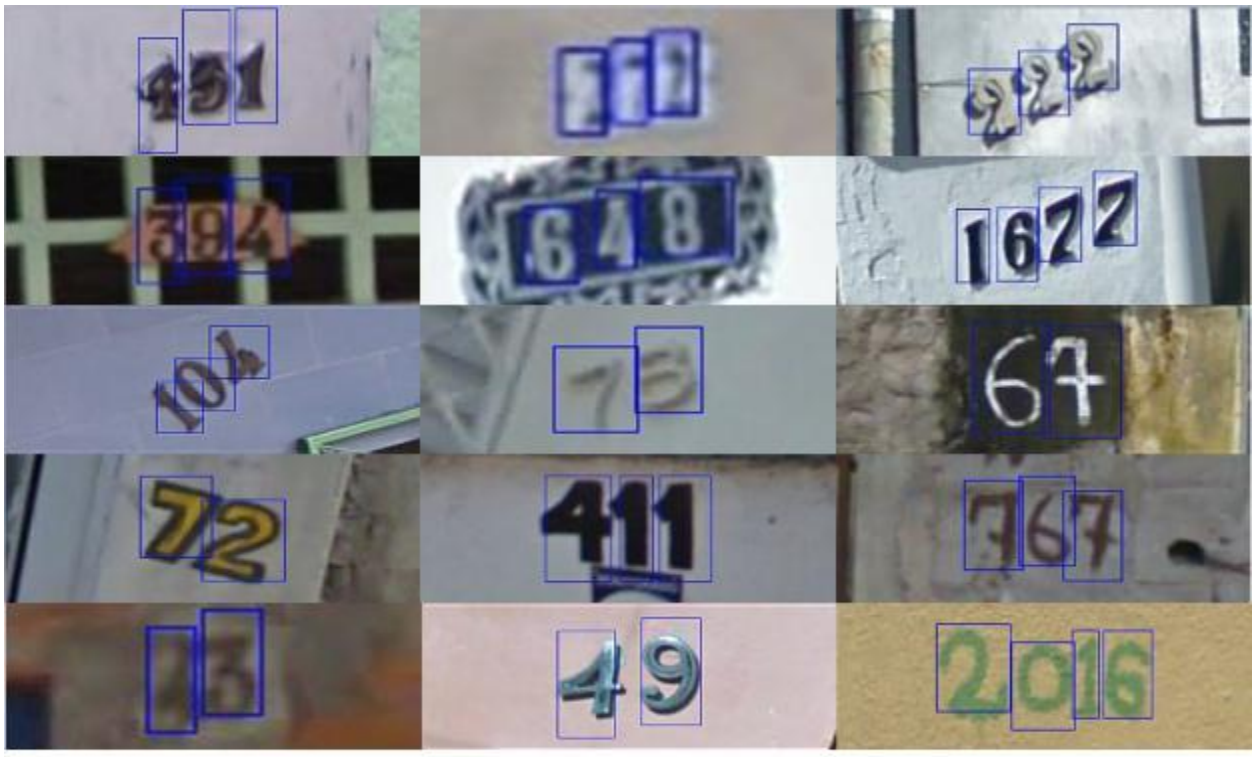

Figure 1: Sample images showing bounding boxes

The .mat file here is a dictionary specifying the sample as well as important attributes of the sample. The important attributes are

1. The digit contained in the email
2. The bounding box of each digit
3. The filename of the actual image.

Loading this file requires the hd5f python library which is free to use. Since the dataset has already been separated into train, test and extra, we also know that the sample has been randomly selected, we have little need to shuffle the data.

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.

Exploratory Visualization

According to the SVHN website, we are the original, variable resolution, color house number images with character level bounding boxes.

Character Height

The character height is distance between the top and the bottom of the bounding box.

Dataset	Mean	Median	Standard Deviation
Train	34.366	30.0	19.378
Test	27.898	24.0	13.459

Table 1: Character height statistics

This shows that the train dataset is potentially a more difficult dataset because the data is more spread out and it contains larger character heights.

Number of Characters

Each image contains a number of characters and here is a visualization to show the number of digits. From our table, it is optimal to train on images with 1 - 5 characters.

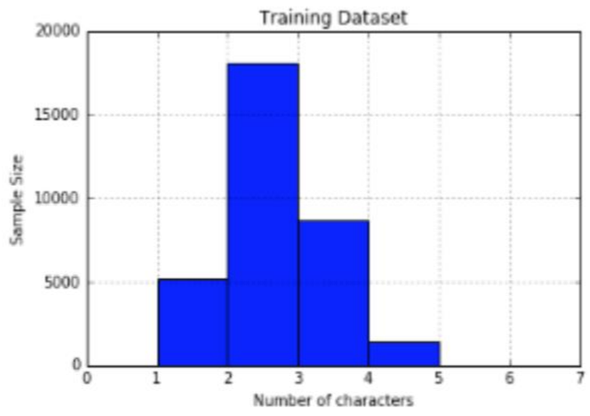

Figure 2: Characters in Training dataset

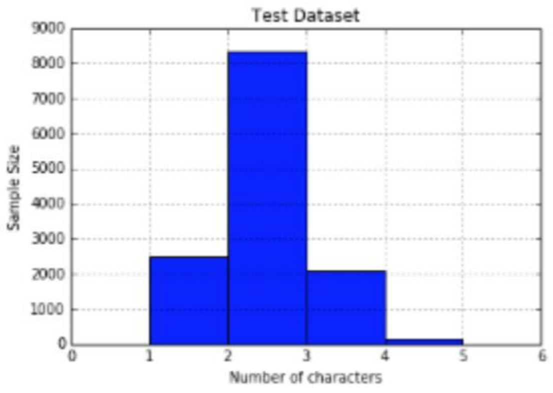

Figure 3: Characters in test dataset

Algorithms and Techniques

I'll be using a Convolution Neural Network (convent) to predict recognize a sequence of digits. This is largely due to the network being able to identify images at different scales. Another advantage here is that a convent is able to directly work on raw pixels.

We'll be using an architecture similar to the one made popular by the tensor flow website to solve the CIFAR-10 dataset.

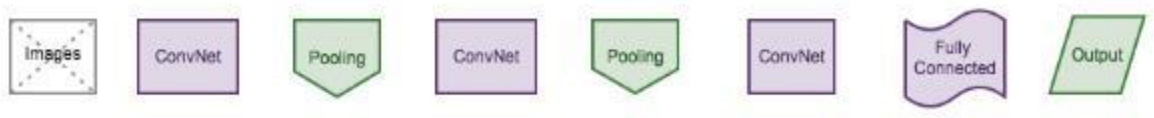

Figure 4: Convolution Neural Network

Images / Input Layer

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

Convent / Convolution Layer

Convents 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 convent. Pooling finds a way to combine all information from a previous convolution into new convolution. For our example, we'll be using carpooling, 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 spatial like 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

The primary benchmark here would be the accuracy of prediction. During our data exploration experiments, our classifier was able to predict single digits with 59% accuracy. Accuracy score is the percentage of correctly predicted data. According to Google's street view data, humans are able to recognize street numbers with 97% accuracy. To create a model that can help, it needs to be able to perform either close to or better than humans, I'd say between 92 and 97%.

III. Methodology

Data Preprocessing

We performed the following steps to preprocess data:

1. Extract information from the digit Strict and save it in a python friendly format.
2. Generate new images using the bounding boxes in the previous image.
3. Resize our new images to 32x32 pixels.

4. Now we generate final training, testing and validation dataset. The validation set is selected by sampling the extra dataset.

Implementation

My implementation is broken down into three steps, and covered in 5 jupyter notebook files.

01-Download and Extraction.ipynb

Here, we pull download and extract the svhn dataset. We also perform checks to make sure that the digitStruct.mat file is present.

02-Exploration.ipynb

Here we explore the dataset and generate visualizations such as a histogram of character count / height.

03-Data Preparation.ipynb

Here we Go through our dataset for train, test and extra, load up the images, crop the bounding boxes and the resize the image to 32x32px. We also save pickle our dataset so it can be reused later.

04-Model Training.ipynb

First, we retrieve the already saved dataset. We also define our model here as well as the accuracy function. For our model, we initialize weights and biases as well as the 5 classifiers that would be used to recognize each digit.

We then train our training dataset, but in batches. Once that is done, we save the trained model to be used in the last notebook 05-Prediction.ipynb

This notebook redefines our model, but instead of training, it loads already saved model. We get some samples of already labelled images and use our model to predict it. These predictions as well as the original samples are shown in figure 5 and 6

Model Training

1. We load up the saved dataset
2. We define our convent (Figure: 4)

3. We define the weights and biases for our 5 logits. The weights are initialized using Xavier initialization - a tensor flow function that ensures that weights are balanced randomly based on the number of neurons
4. We define our Loss and accuracy function
5. We train our model and log important information like accuracy of the validation set and the loss.
6. Save the trained model to disk so it can be loaded again or exported

Our Convolution model

- C1: convolutional layer, batch_size x 28 x 28 x 16, convolution size: 5 x 5 x 1 x 16
- S2: sub-sampling 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: sub-sampling 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: fully-connected layer, weight size: 64 x 16
- Output layer, weightsize: 16 x 10

That brings it up to 7 layers.

To train, we read the already 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. One step further would be to save the model and load the model so it can be used in other applications such as an android app.

Refinement

The initial model produced an accuracy of 82.1% with just 30000 steps. I performed the following improvements to improve the accuracy score

1. I added a dropout layer to the network just before fully connected layer. This was to prevent over fitting. It works by randomly dropping weights from the model. I set the keep probability to 0.9375.

2. I also changed the learning rate to exponential decay instead of keeping it constant so that it learns fast initially and then slowly over time.

After implementing these, my model now predicts with 85.9% accuracy just after 30000 training examples. It should perform more with more training examples

IV. Results

Model Evaluation and Validation

At the end of my training, I was able to achieve the following:

Minibatch loss: 1.077957

Minibatch accuracy: 93.8%

Validation accuracy: 75.3%

Test accuracy: 85.9%

According to our benchmark, this model produced much more accurate predictions when compared with the base classifier, although not as good as humans do.

The final model is made up of the following:

- Weights are initialized using Xavier Initializer
- We implemented Max pooling to the hidden layers
- We implemented convolutions for depths at 16, 32 and 64
- We created 5 classifiers / logits
- We implemented learning rate decay at 0.05
- We used AdagradOptimizer as our optimizer
- We introduced dropout just before the fully connected layer with 0.9375 keep probability
- We used accuracy score as our benchmark

Justification

Our benchmark of human recognition is 97% while that of a naive classifier was 59%. Our model was able to achieve 85.9% accuracy while training on 30000. 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. According to the paper written by on this model, one can achieve over 97% accuracy with this model.

V. Conclusion

Free-Form Visualization

To test how well our model is working, we created random images and used it for our prediction. Here is the actual images and their labels:

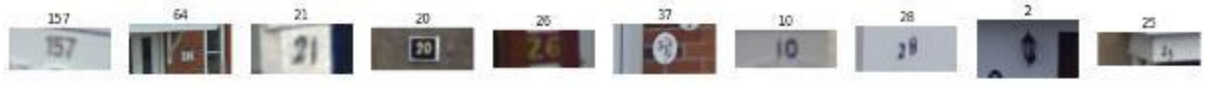

Figure 5: Sample Images for prediction

These images were then passed through our model and here is what our model predicted the values to be

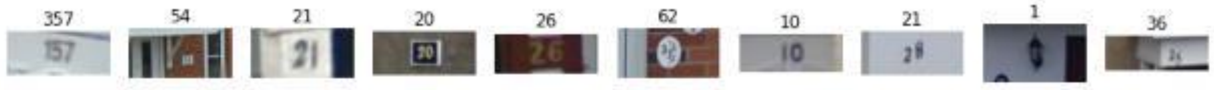

Figure 6: Sample Images Predicted

We can see that our model did a pretty good job at predicting data and it did so with a training set of just over 30000. If we continue to train with the larger set of 200,000 images, we will achieve even better results, however, my computer might take a really long time to do that, since i am training on a CPU.

Reflection

Deep learning is an interesting field of machine learning that can be applied to many exciting problems. The problem we have applied it to is housing number digit recognition.

While it is capable of solving many problems, the part that I find most challenging is the specialized compute resource needed to get good results. GPUs are becoming cheaper and I believe that every device would soon be able to train deep learning models, or at least retrain their models based on new information.

The initial solution was to synthetically create a new dataset by combining digits together from either MNIST or SVHN 32x32px dataset. My earlier experiments worked when predicting one character and 5 characters in a sequence but performed poorly on a real dataset. So I used the bounding boxes instead.

Another thing here was to resize images into 32x32px images. My earlier model used 800x600px images to simulate what a phone screen might look like. I was unable to train it because of the sheer amount of memory needed to process that dataset. I rented a 60GB virtual machine in the cloud and I have over 30GB of memory used up on the first 2000 images using this architecture.

I also converted images to grayscale by averaging the pixels. This was to further reduce the memory footprint. Some other scholars believe there is a certain ratio of Green, Red and Blue that makes an image appear closer to human perspective, however, I did not explore that option.

References

- http://ufldl.stanford.edu/housenumbers
- http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42241.pdf
- https://en.wikipedia.org/wiki/Convolutional_neural_network
- https://www.tensorflow.org/versions/r0.10/tutorials/deep_cnn/index.html
- http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow