Capstone Project

Machine Learning Engineer Nanodegree

Himanshu Dongre
August 16th, 2016

I. Definition

Project Overview

Recognizing multi digit numbers from street level photographs is an important aspect of modern day digital map making. Consider for example being able to automatically pin point an address with high degree of accuracy just with street level images of that building.

In this project, I created an application which uses public Street View House Numbers(SVHN) dataset to train a Convoluted Neural Network model with multiple layers to understand and be able to detect with high accuracy, the house address numbers with just street level photographs.

Problem Statement

The goal of the project is to create an application which when provided with a street level image containing a house number, be able to detect that number from just image pixels.

I propose the following steps to tackle the problem discussed above:

1. Download and extract the entire SVHN multi dataset on the local system.
2. Crop images to bounded region to find digits.
3. Find Images with more than 5 digits and delete it from training set.
4. Random shuffle valid dataset and save in a pickle file.
5. Use the preprocessed dataset pickle file and train a 7 layer Convoluted Neural Network.
6. Use test data to check for accuracy of the trained model to detect number from street house number image.

Metrics

As the purpose of this number detection is to ultimately use it for auto geotagging of buildings on a digital map, both true positives and true negatives matter as we need to be as accurate while classifying as possible. As we are trying to detect all the digits of the number simultaneously, even a single false negative or false positive will result in a wrong number. Thus accuracy seems to be the prime candidate to be used as a metric in this scenario.

$$Accuracy = \frac{true\ positives + true\ negatives}{dataset\ size}$$

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.

Features:

1. 10 classes, 1 for each digit. Digit `'1'` has label 1, `'9'` has label 9 and `'0'` has label 10.
2. 73257 digits for training, 26032 digits for testing, and 531131 additional, somewhat less difficult samples, to use as extra training data
3. Comes in two formats:
   - Original images with character level bounding boxes.
   - MNIST-like 32-by-32 images centered around a single character (many of the images do contain some distractors at the sides).

For our project we are using the full number format dataset:

Full Numbers: train.tar.gz, test.tar.gz , extra.tar.gz

Fig.1: Some samples from Full number format of datset with bounded boxes for digits.

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 original 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

As we can clearly see in Fig.1 above that in SVHN dataset most of the house number signs are printed signs and thus are easier to read. However, they have large variations in their font, size and colors making the detection difficult.

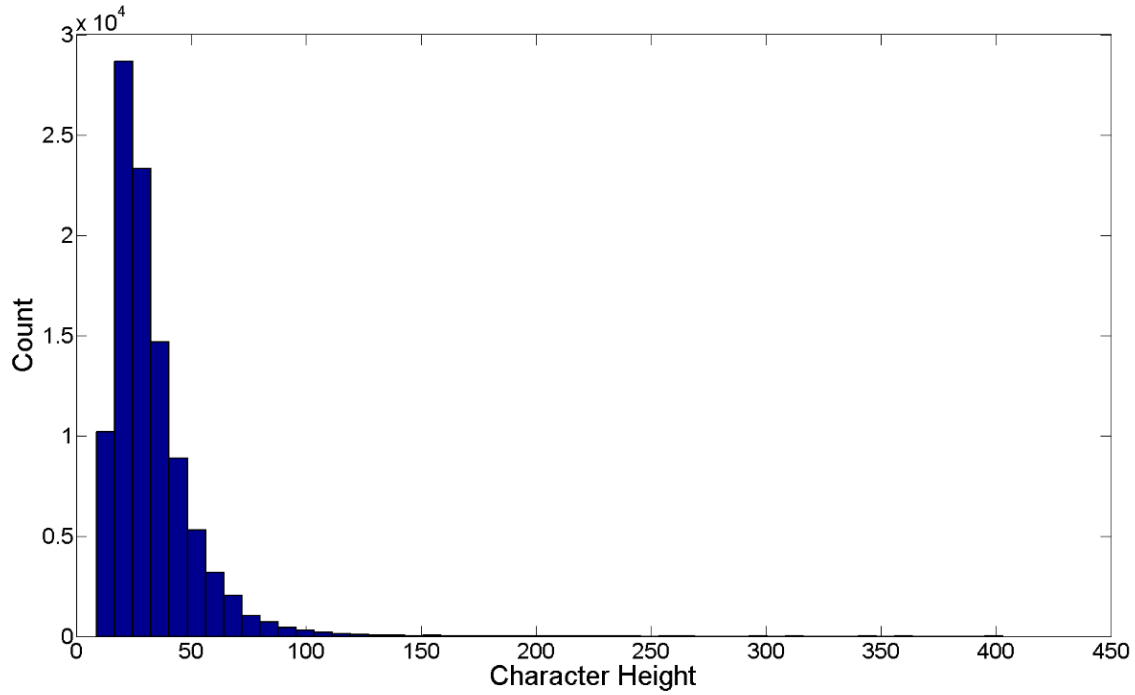

Fig.2: Histogram of SVHN characters height in the original image. Resolution variation is large. (Median: 28 pixels. Max: 403 pixels. Min: 9 pixels.)

Fig.2 above shows large variation in the character height as measured by the height of the bounding box in original street view dataset. This suggests that size of all characters in the dataset, their placement and character resolution is not uniform across the dataset. This further adds to the difficulty of making a correct house number detection.

Algorithms and Techniques

A Convoluted Neural Network(CNN) is generally considered as the go to algorithm for most image recognition problems. This is due to their ability to directly work on the raw pixels of image.

For our problem of house number detection, we will be using the following CNN architecture:

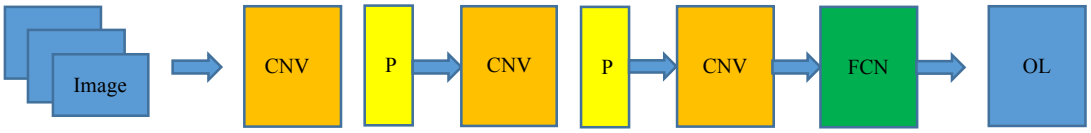

Fig.3: CNN architecture for SVHN dataset house number detection

Convolution Layer(CNV): In a CNN network a convolution layer is an input data filtering layer which consists of learnable filter parameters. These learnable parameters have a small receptive field but they extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter which is also called as the feature map. As a result, the network learns filters that activate when they see some specific type of feature at some spatial position in the input.

Sub-sampling or Pooling(P): Pooling is another important aspect in a CNN. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum. The intuition is that once a feature has been found, its exact location isn't as important as its rough location relative to other features. The function of the pooling layer is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. It is common to periodically insert a pooling layer in-between successive conv layers in a CNN architecture. The pooling operation provides a form of translation invariance.

Rectified Linear Units (ReLU): These are used in the network to increase non linearity of the decision function without affecting the receptive fields of the convolution layer.

Fully Connected Layer(FCN): A FCN layer is used in a CNN to provide a high level reasoning to the work done by other CNV and pooling layers. This layer has connection to all the activations from previous layers.

Output Layer(OL): Output layer is the final layer in a CNN which provides with the logits for the 5 digits detected by the network.

Benchmark

The main aim of this project is to detect house numbers to automatically geotag a building in a map. Currently this work is mostly done manually by human operators which has an accuracy of 98%. Since we want to automate the process, the bench mark to achieve should be 98% or more. However, for creating such state of the art architecture and train it I would require resources which are currently not at my disposal. Therefore, comparing it with other approaches for SVHN, I have decided to target an accuracy of around 90% for SVHN dataset by using my MacBook CPU for processing.

III. Methodology

Data Preprocessing

In svhn_preprocess notebook, following preprocessing is performed on the dataset:

1. We extract all the data from Digit struct file and save all the data to a python dictionary for test, train and extra data.
2. Now we crop the images to the region of the bounded box information that we got from Digit Struct file for images in test, train and extra dataset.
3. Now we generate 32x32 size images from our dataset for training, testing and validation.
4. We also find and delete all those data points which has more than 5 digits in the image.
5. Now we generate final training, testing and validation dataset by concatenating datapoints from extra dataset and then save this in a pickle file to be used by svhn_cnn notebook.

Implementation

For implementing our model, we have used the pickle file svhn_multi which we created by preprocessing the data from the original SVHN dataset. In svhn_cnn notebook, following steps are implemented:

1.We first load the data from svhn_multi pickle file into training, testing and validation dataset.

2.Now we construct the following layered model for a CNN as described in Fig.3 above:

7-layer CNN:

1. C1: convolutional layer with batch_size x 28 x 28 x 16 and convolution size: 5 x 5 x 1 x 16
2. S2: sub-sampling layer with batch_size x 14 x 14 x 16
3. C3: convolutional layer with batch_size x 10 x 10 x 32 and convolution size: 5 x 5 x 16 x 32
4. S4: sub-sampling layer with batch_size x 5 x 5 x 32
5. C5: convolutional layer with batch_size x 1 x 1 x 64 and convolution size: 5 x 5 x 32 x 64
6. F6: fully-connected layer with weight size: 64 x 16

7. Output layer with weight size: 16 x 10

   Note: A few ReLU layers are also added after each layer to add more non linearity to the decision making process.

3.We now define the weights and bias for the 5 logits which we are trying to detect simultaneously. Weights are initialized randomly using Xavier initialization which keeps the weights in the right range. It automatically scales the initialization based on number of output and input neurons.

4.We also define accuracy and loss function for our model as follows:

Accuracy:

```
def accuracy(predictions, labels):

  return (100.0 * np.sum(np.argmax(predictions, 2).T == labels) / predi
ctions.shape[1] / predictions.shape[0])
```

Loss Function:

```
loss = tf.reduce_mean(
tf.nn.sparse_softmax_cross_entropy_with_logits(logits1, tf_train_labels
[:,1])) +\
tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(logits2,
tf_train_labels[:,2])) +\
tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(logits3,
tf_train_labels[:,3])) +\
tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(logits4,
tf_train_labels[:,4])) +\
tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(logits5,
tf_train_labels[:,5]))
```

5.We now train the network and log the accuracy, loss and validation accuracy in steps of 500.

6.Initially we used a static learning rate of 0.01 but later on switched to exponential decay learning rate with an initial learning rate of 0.05 which decays every 10000 steps with a base of 0.95.

7.Used Adagrad Optimizer to minimize loss.

8.We stop learning when we reach adequate accuracy level for the test dataset and we save the hyper parameters in cnn_multi checkpoint file so that it can be loaded later when we need to perform detection without training the model again.

Refinement

The initial model produced an accuracy of 89% with just 15000 steps. Its a great starting point and certainly after a few hours of training the accuracy will definitely reach my benchmark of 90%. However, I further made some simple improvements to further increase the accuracy with few number of learning steps.

1. Added a dropout layer to the network after the third convolution layer just before fully connected layer, which randomly drops weights from the network with a keep probability of 0.9375 to add more redundancy to the network. This allows the network to become more robust and prevents overfitting.
2. Introduced exponential decay to learning rate instead of keeping it constant. This helps the network to take bigger steps at first so that it learns fast but overtime as we move closer to global minimum, take smaller noisier steps.

With these changes, the model is now able to produce accuracy of 92.9% on test set with 15000 steps. Since there are 230070 images in training set and about 13068 images in test set, the model is expected to improve further if it is trained for longer duration.

IV. Results

Model Evaluation and Validation

The final model is able to reach accuracy of 92.9% on test set which is above the benchmark set for the project. I am saving the hyper parameters in a checkpoint file so that it can be restored later to continue training or to use it for detection over new images. Use of dropout ensures that the model is robust and does learn a redundant generalized model which can do well on most data. The model is tested over a wide range of input from the test dataset and generalizes well.

Fig.4: Some labeled output from the model

It can be seen from Fig. 4 that the model is able to predict the data clearly for most of the images. However, it still gives incorrect output when the images are blurry or plagued by other noise such as paint.

Justification

The model performs well above the set benchmark of 90%. Due to time constraints its trained for a relatively less duration. Its expected that the accuracy will increase further with longer training durations. With better hardware and GPU support the model can be further tweaked to train and run faster.

The purpose of the project was to develop a system which can replace manual geotagging of house numbers by human operators. Studies show that the accuracy of human operators in this task is about 98%. So the model is still a long way from being used to replace human operators. However, the results from the project show that the technique used is quite effective and may reach the required accuracy with specialized hardware and further tweaks to the architecture.

V. Conclusion

Free-Form Visualization

Fig.5: Some more examples from the model output

It can be clearly seen that the model produces correct output for most images. However, the detection fails when one or more of the following case occurs:

1. Image is blurry
2. Image has unwanted noise which makes it difficult to detect even for human operator
3. Number in image are far away or at an angle making the detection difficult.

Reflection

The project is divided into 2 python notebooks:

1. Svhn_preprocess
2. Svhn_cnn

In svhn_preprocess, we perform following steps:

1. We first download the dataset to local machine if its not already available.
2. Once dataset is available on local machine we extract the images into individual folders.
3. We also extract the DataStruct file which contains the co-ordinates for the bounded boxes which identify our numbers in the image dataset.

4. We now crop our images to only contain the numbers as specified by the bounded boxes and we remove any images which have more than 5 digits for detection simplicity.
5. We now save the processed dataset in a pickle file.

In schn_cnn, we perform following steps:

1. We first load the preprocessed dataset from the pickle file into respective test, train and validation set.
2. We now define weights and biases to be used for 5 logits that will be detected simultaneously.
3. We now define our 7 layered CNN architecture which will output the required logits.
4. We not define accuracy and loss function for the problem.
5. We run our model for a few thousand steps and check the mini batch accuracy and validation accuracy for every batch of 500 and once the training is complete we check the test set accuracy.
6. We now save the currently learned hyper parameters in a checkpoint file so that it can be used again for detection without the need of retraining the network.

One interesting aspect of the project was to find out how well the optimization tricks like dropout and exponential learning rate decay perform on real data.

One difficult aspect of the project was to choose appropriate architecture for the problem. Since there are many ways the architecture can be implemented its very difficult to understand why a particular architecture will work best for particular type of data.

The model implemented here is relatively simple but does the job very well and is quite robust, however its still requires a lot of work to make the model perform equivalent or better than a human operator.

Improvement

We can see from Fig.4 that the model gives incorrect output for images which are blurry. To tackle this problem, we can use a local contrast normalization step in preprocessing to increase the contrast of the images. This can help us in detecting numbers efficiently. Also GPU support can be added to further speedup the process of training.

References:

- 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*.
- https://en.wikipedia.org/wiki/Convolutional_neural_network
- http://ufldl.stanford.edu/housenumbers (dataset link)
- http://www.iapr-tc11.org/mediawiki/index.php/The_Street_View_House_Numbers_(SVHN)_Dataset
- http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42241.pdf
- https://arxiv.org/pdf/1204.3968.pdf
- Udacity forum