Capstone Project

Machine Learning Engineer Nanodegree

Hania El Ayoubi
July 3rd, 2016

Definition

Project Overview

For those gifted with vision, it is easy to take for granted the complexity and sophistication that the eyes and brain work together to recognise the objects around us. Objects come in various shapes and colours, may be partially occluded, with various orientations, yet the gifted with vision are able to recognise them with ease that does not reflect the complexity involved in the process.

This complexity is well understood by researchers in the field of computer vision, who for decades have strived to allow computers to "see". What we perceive as an image, the computer can only perceive as a bunch of electric 0 and 1 signals, and the challenge in computer vision is to build algorithms that can withstand the complexities involved to allow computers to recognise objects. There have been competitions such as ImageNet[1] to test these approaches and even images generated from the CAPTCHA program[2] to purportedly tell humans apart from bots.

The Street View House Number (SVHN)[3] data set is a large set of labeled images obtained from house numbers. As with any real life object, the house numbers come in different colours, styles, alignment and orientation. This project aims to use some of the cutting edge approaches, namely deep learning, and use the SVHN dataset to train and test how well the deep learning model is able to recognise previously unseen SVHN images.

[1] http://image-net.org/challenges/LSVRC/2016/index
[2] http://www.captcha.net/
[3] http://ufldl.stanford.edu/housenumbers/

Problem Statement

This project aims to use deep learning to recognise digits in the SVHN dataset up to three digits of length[4]. The tasks involved are as follows:

1. Data exploration of SVHN, to understand the characteristics of the training set and the limitations in the data.
2. Data preprocessing of SVHN, to transform all images so they're of the same cropped dimension, as deep learning pipelines depend on fixing the number of features[5] presented to them.
3. Data augmentation, to translate the images and crop them randomly while maintaining the original digits within them, in order to increase the input data available to the algorithm.
4. Defining the deep learning optimisation model in the Tenserflow framework[6].
5. Training the deep learning model on the augmented data.
6. Testing the model against the test set of SVHN.

The anticipated solution is a model that can recognise digits (and in the case of sequences shorter than 3 digits the *absence* of a digit) in the previously "unseen" test dataset of SVHN.

Metrics

The primary metric used is the accuracy of prediction of each digit (or absence of a digit[7]). Following the training phase will be the test phase. The test set of SVHN is in the thousands, and to make testing the model practical with the memory limitations on my machine, the test set will be broken in batches and an average of accuracy over each digit will be calculated as the overall performance of the model.

---

[4] The SVHN dataset contains images with digit sequences longer than 3 digits, however, to make for a practical training duration, I chose to restrict the dataset to 3 digits or less.

[5] An image with dimensions 64 x 64 pixels and RGB encoding will have 64 x 64 x 3 = 12288 features, in the machine learning sense of the word.

[6] https://www.tensorflow.org/

[7] In addition to the ten digit classes, a new eleventh class will be added to stand in for a blank digit.

Analysis

Data Exploration

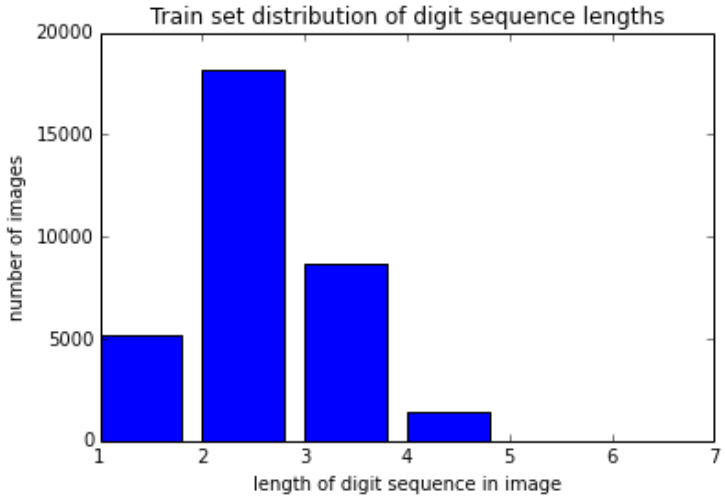
Here are a few SVHN images:

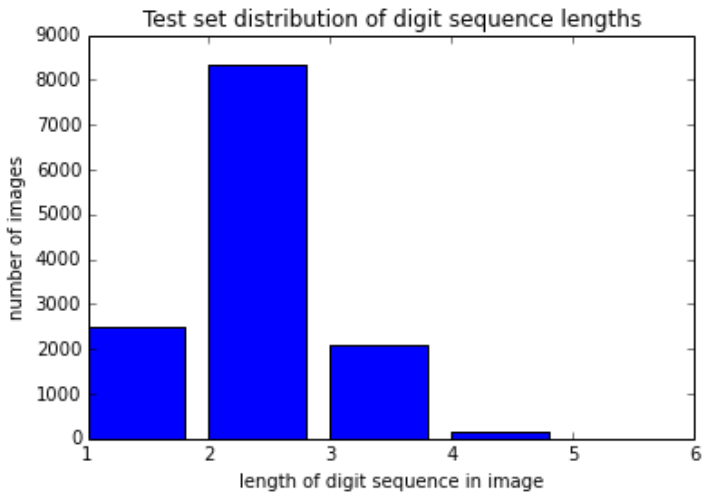
A few characteristics of the SVHN dataset that make it so challenging to use can be observed from these images:

1) The images have multiple colours, and the digits also come in various colours
2) There are other objects such as a frame or other characters acting as noise in the image
3) The digits to be recognised come in various orientations, as in image 180 above
4) The digits across images have varying sizes and fonts
5) The dimensions of each input image is varying
6) The digits are sometimes hard to guess even for the human eye

Exploratory Visualization

The SVHN dataset is divided into training and test sets. Breaking the datasets down into digit sequence length in each image, there is a similar distribution in the train and test sets when the digit sequence lengths is less than or equal to three digits. Beyond that, images with four digits or more become scarce, hence this project will focus on training with images of digit sequence length three or less.



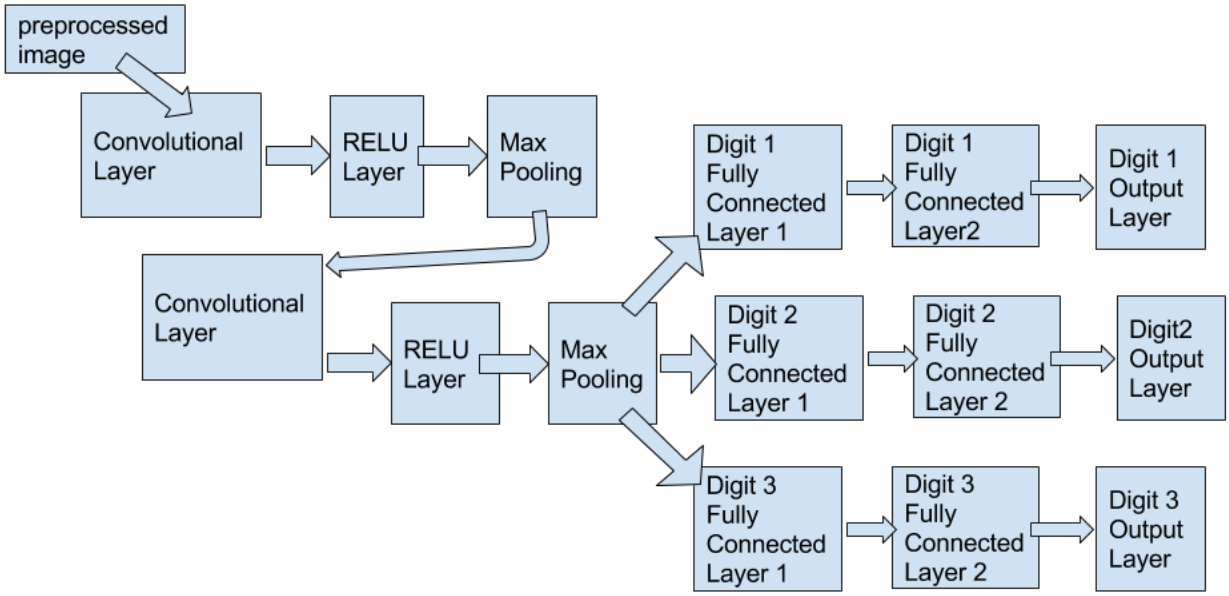


A bar chart was chosen to represent this because there is set of categories (1, 2, 3-digit sequences, etc). This is an important visualisation because the type of data we feed to the deep learning algorithm will influence the training, hence we should be aware of any skews in the train and test data set. Luckily, the skew in the training set toward 2-digit sequences is matched in the test set.

Algorithms and Techniques

The model to recognise digits in SVHN images in this project is a convolutional neural network (CNN). Deep neural networks are valuable in image recognition due to the feature selection built in to the model in the convolutional and max pooling layers. Also, the images have dimension 64 x 64 x 3, so with 12288 features, having a convolutional network architecture is superior to a regular neural network because the convolutional network is more powerful at modeling the image while using fewer parameters per layer. With a regular neural network, a large number of neurons would be needed each layer, which is computationally expensive and is thus inhibitive.

The CNN used has the following structure:



Benchmark

There are several published results for the SVHN dataset[8]. The paper used to guide this project was the one by (Goodfellow et al, 2014)[9]. In that paper, they were able to obtain 98% accuracy. Given the difficulty of the SVHN dataset, this project will gauge performance as follows:

1) Accuracy of digit 1 in the test set, this should be well above pure chance, i.e. 10% to be able to conclude that the model did learn
2) Accuracy of digit 2 in the test set, this includes the ability to detect the absence of a digit
3) Accuracy of digit 3 in the test set, this also includes the ability to detect the absence of a digit

[8] See http://goo.gl/4qeP8w

[9] Goodfellow, I., Bulatov, Y., Ibarz, J., Arnoud, S. , Shet, V. (2014) Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks

Methodology

Data Preprocessing

1) Decoding the Data Labels

   The SVHN data is made up of images and a single label file in Matlab's v7.3 dat format. As this format was not easily readable into python, I installed Matlab to convert it from v7.3 to v7, which could be read using the scipy's io module. The training and test v7 files are stored with the project files for ease of use.

2) Data Augmentation and Scaling

   In order to augment the training data set to provide more training samples to the machine learning algorithm, each image is cropped so that the new image is 60% of the original, and then translated to maintain the "outer bounding box" of the digits[10].

   Since the deep learning pipeline expects images of a fixed dimension, the output of the pre-processing step is an image scaled to dimensions 64x64x3.

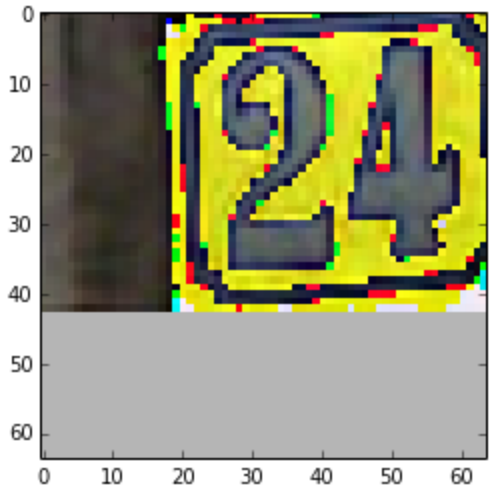
3) Zero-mean

   For each RGB channel in each image, the mean pixel value is subtracted to give a zero mean to the image.

   Here is a sample image before preprocessing:

   And here it is after preprocessing:

[10] The SVHN dataset contains dimensions to bounding boxes within its labels, which are helpful when translating the image to make sure the resulting image does not lose digit information.



Implementation

This project used Convolutional Neural Networks implemented in the Tensorflow framework. The model is comprised of the following:

1) Input layer of 64x64x3 = 12288 features, holding the preprocessed pixels of the 64x64 image in the three RGB channels.
2) Convolutional layer with stride 1 and padding=SAME, which means that the output feature map has the same spatial dimensions as the input feature map.
3) RELU layer to apply the activation function.
4) Max pooling layer with stride 2.
5) Dropout of 0.5 to prevent overfitting and allow training for longer epochs.
6) Convolutional layer with stride 1 and padding=SAME.
7) RELU layer.
8) Max pooling layer with stride 2.
9) Dropout of 0.5.
10) For each digit, there are two fully connected layers, each with RELU activation.
11) Finally there is the output layer to output the logits.

The training used a mini-batch stochastic gradient descent, with batch size 128. A list of all images was generated, and a random offset into that list was used to create each offset.

Refinement

Convolutional networks are very powerful, but they come with a large set of hyper parameters. Here are the ones that were experimented with. The experimentation was done to determine the variant of each hyperparameter that gave the best convergence within an hour. Note that the actual training took 48 consecutive hours.

The parameter selection was heavily influenced by the architecture of the machine selected to run the training. The machine was a g2.2xlarge AWS EC2 virtual machine, with 1 NVIDIA GPU with 4GB of video memory and 15 GB RAM.

Here are the parameter refinements over the course of the project:

1) Parameter initialisation

   Rather than initialise the model to small random values, the weights and biases were initialised to a truncated normal distribution with standard deviation of

$$\frac{1}{\sqrt{size\ of\ first\ dimension\ of\ the\ layer's\ volume}}$$

2) Optimisation algorithm

   The Gradient Descent and AdamOptimizer were compared, and the latter was found to have better characteristics for convergence.

3) Starting learning rate

   Several learning rates were tried: 0.1, 0.001, 0.0001, 0.00001.
   0.0001 was the value selected as the best, which although was a slow learning rate but it prevented overshooting and oscillation around the minimum.

4) Dropout

   Dropout was not immediately visibly beneficial, because it substantially increases the required learning epochs, but it was used to prevent overfitting and build redundancy within the model's neurons.

5) Regularisation

   With the large number of weights, regularisation was not found to be useful in this case, and given that dropout was used, no regularisation was implemented.

6) Depth of each convolutional layer

   This is by far the most tricky parameter to try out.

7) Strides selected for convolutional layers and max pooling

For this parameter, I settled on using a stride of 1 for the convolutional layer and 2 for max pooling. I didn't want to use larger strides so as not to lose information, and for max pooling, a stride of 1 increased the memory requirements of the model to the point after which it could not run on my machine.

Results

Model Evaluation and Validation

The model described was run in training epochs of mini-batch stochastic gradient descent for 48 hours and the final model parameters were saved via Tensorflow. To evaluate the model, the entire test set with digit sequences of length three or less were used.

To introduce robustness into the model and the testing phase, the preprocessing step of cropping and sliding the input images was applied to both the train and test set, to force the model to withstand such changes. Also, while training, a random offset into the training set was used for each epoch.

The test set is too large to fit in memory to assess in one batch, so instead, it was broken into 20 batches and the accuracy of each digit detection was calculated as an average over these 20 batches.

The accuracy results obtained for this model was as follows:

1) Digit 1: 74.8%
2) Digit 2: 67.4%
3) Digit 3: 89.5%

Justification

In the test set, 12920 in total, recall that the breakdown of images is as follows:

1) 19% with 1 digit
2) 65% with 2 digits
3) 16% with 3 digits

Also recall that the reference paper by Goodfellow et al, 2014 reported 98% accuracy.

The relatively high accuracy of digit 3 represents the model's relative success at detecting an empty digit, as 84% of the test images do not have a third digit. For digit 1, the accuracy of 74.8% is well

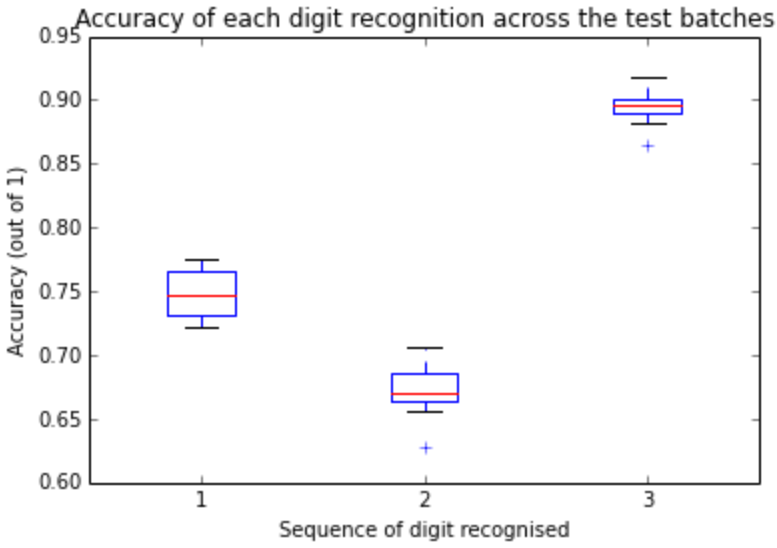
above the 10% of getting the first digit right due to chance. The digit 2 accuracy is well below expected, but it is also a reflection of the difficulty of the SVHN dataset[11].

Also worth noting is that the model in Goodfellow et al, 2014 ran for 6 days, while my model ran on a relatively small Amazon EC2 g2.2xlarge for 2 days. It would have been impractical and quite costly to continue training for over 2 days.

Conclusion

Free-Form Visualization

The following diagram summarises the results in one diagram:



This chart was chosen because it summarises all the test results in a succinct fashion, and highlights the wide variability of accuracy results between one batch and the next.

Reflection

Deep learning has a lot of hyperparameters and the length of time needed to train makes working with a tough dataset such as SVHN truly challenging. Through the development of the convolutional network model, I developed the intuition that guided the hyperparameter selection. In attempting this project, I worked with MNIST and generated images with multiple digits. However, I found that a model trained for MNIST multiple digits was of no immediate value for

[11] Note that my same model framework was used for the MNIST set of handwritten digits, http://yann.lecun.com/exdb/mnist/, and MNIST-generated sequences of multiple digits per image, and obtained an accuracy of over 90% for each of the three digits.

SVHN dataset given even the training error. The framework was the same, but the training and data preprocessing was very specific to the SVHN dataset.

While the accuracy results of my model are inferior to the Goodfellow et al, 2014 result, it is still much higher than chance and as such shows indication of learning within the model. The limited training set showed signs of overfitting after 48 hours of training, and perhaps having more data would have helped. Having cleaner data like MNIST would have greatly boosted the results, and this showed me just how difficult of a problem vision is.

Improvement

One idea from the Goodfellow et al, 2014 paper that I wanted to implement was adding a length parameter to the Tensorflow graph, and making the model recognise correctly the length of the sequence, such that the resulting optimisation calculation at each epoch took into account the length of the digit sequence and did not take into account digit logits for digits greater than the predicted length. However, given the state of Tensorflow v0.8 at the time of model building, that was not possible, so in the future, when subsetting by boolean arrays becomes possible, I would like to revisit the model and implement length prediction fully.