Street-View House Numbers Dataset

Giannis Konstantinidis (BAPT1503)

Stamatis Pitsios (BAPT1502)

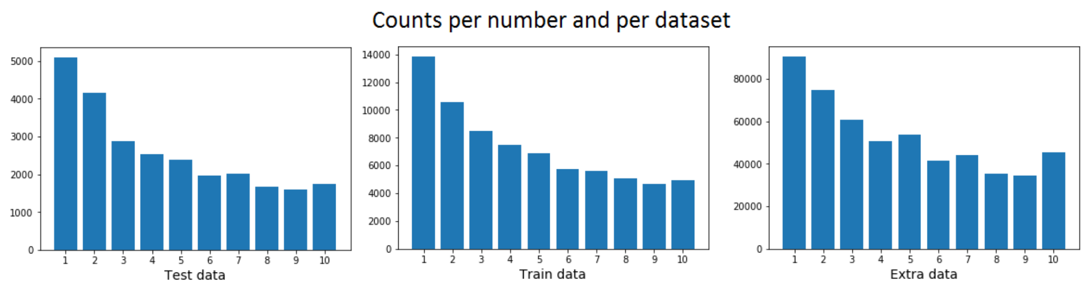
Lefteris Vazaios (BAPT1516)

Contents

Introduction

The aim of this study is to perform a deep learning analysis on image data, measure and compare results and describe our findings. We are looking to train a model that would recognize as accurately as possible each of the images available.

The data consist of 32x32 colored pictures of house numbers and originate from Google's Street View. Each image represents a single one-digit number. In total, 73257 digits were available for training, 26032 for testing and 531131 "easier to guess" digits for further testing. The overall images were of various colors, sizes and orientations, usually containing more than one digits. Those have been already cropped and stored in MATLAB files (.mat), resulting in the dataset we will be using. The images were represented as vectors of numbers representing the RGB (red – green – blue) values for each pixel. Those values range anywhere between 0 and 255. Since the data concern house numbers, it is rather obvious that the smaller the number, the more likely it is to appear. Nevertheless, a simple histogram showing the frequency of each number in our images follows:



The data, along with a short description can be accessed from the following link:

http://ufldl.stanford.edu/housenumbers/

For the purposes of our analysis we will be using Python, focusing on a specific library specializing in deep learning, Tensorflow. The general technique or family of algorithms we will be using is neural networks, with notable examples being CNN and MLP.

The methodology used, described extensively below, can be briefly summarized to preprocessing the data, applying several algorithms (experimenting with their parameters and techniques suggested by similar studies), comparing results and extracting conclusions.

Preprocessing

As it has been previously mentioned, the data came with a certain degree of preprocessing already implemented. More specifically, the images had already been cropped so as to contain – whenever possible – one digit per each 32x32 image. However, it was impossible to entirely avoid a certain degree of noise occurring in several of the images available. Furthermore, the mere variety of colors and brightness in the training dataset could be the cause for more problems and lower accuracy for our algorithms.

To counteract those issues, we experimented with two basic methods, also proposed by relevant research on the topic. First off, a normalization technique was implemented in order to reduce the range of potential colors in each image (essentially their intensity, not the colors themselves). Out of several alternatives proposed by fellow researchers, we mainly experimented with three:

- Dividing each 0-256 value by 128 and then subtracting 1. This serves to reduce the range of values for each image to [-1, 1].
- Actual normalization, i.e. subtracting the global mean from each value and then dividing by the global standard deviation.
- Batch normalization, a built-in Tensorflow function, essentially implementing what we attempted to compute manually.

The reasoning behind this process is to contain the range of values so that the weights applied in the later stages of the neural network are not overly influential for some dimension and less so for others.

Besides the normalization itself, gray scaling was also implemented, since we are not concerned about the color of the image, just its shape (i.e. the number shown). However, this was not always beneficial to the overall result, in most cases even worsening the classification accuracy.

Hardware Available

The hardware used were two different machines. This was done in order to calculate the speedup that the usage of a GPU offered. In the end it was significant, reducing a 20-minute execution on the CPU to under a minute using a GPU. Therefore, we believe that it is vital to invest in a compatible GPU even for relatively small tasks. In order to load and process the dataset 8GB of RAM was used and the 3GB of the GPU was fully utilized.

- The first machine run Debian 8, had no GPU acceleration and used a VM to perform the computations. Specifications:

  CPU Core i5 4570 @3.6GHz (4 Cores/4 Threads)

  RAM 16GB ddr3

  GPU -

- The second machine used was a gpu accelerated one with cuda enabled and was running natively Ubuntu 16.4 (without a VM). Specifications:

  CPU Core i7 4720hq @3.6GHz (4 Cores/8Threads)

  RAM 32GB ddr3

  GPU Nvidia 970M 3GB Vram

Both systems used SSD (flash) storage, so IO bottleneck was not an issue.

Algorithms Used

The algorithms implemented for this study were Multilayer Perceptron (MLP) and Convolutional Neural Networks (CNN).

The MLP algorithm consists of an input layer (in this case our 32x32 images), one or more hidden layers and an output layer. Each layer consists of several neurons, with the only constants being the input (32x32 = 1024 nodes) and the output (10 nodes, as many as the classes we need to predict). Each hidden layer performs a transformation of the data by assigning weights to the inputs and applying an activation function (non-linear). At the output stage, the algorithm ‘checks’ if it has correctly classified the image and if not, it back-propagates any errors and the weights in each layer get reassigned until the prediction for the input in question is finally correct.

The CNN algorithm consists of 4 primary stages: convolution, non-linearity, pooling and classification. During convolution, features are extracted from the input image by using a sliding filter (smaller dimensions than the original) and getting a dot product (result of matrix multiplication) called a feature map. Different filters can be applied and feature maps of various sizes can be produced. After the convolution process, a non-linear function is again applied to each pixel (much like MLP). In the pooling step, the size of the feature map is reduced even further by applying a sliding window filter to it, extracting the maximum (or sometimes average) of all values present in the window each time. The last pooling layer is the input of the fully connected layer, the final stage of the process, which is basically a simple MLP. Back-propagation of errors occurs here as well, so that the model properly corrects its mistakes.

Implementation

The implementation we opted for (as it presented the best overall results) was the CNN algorithm. Several different versions were experimented with but one was eventually selected as the optimal (mostly in terms of overall performance). The procedure followed in this approach, including any parameters and transformations, is presented below:

- The training data selected for the model was expanded to include the test set with the easier to recognize cases. Relevant research work suggested that doing so would considerably improve the model’s accuracy, as we also later discovered. The data were read directly as MATLAB files using Python’s library scipy.
- The input (i.e. the original training set and the “easy-case” test set) was then subjected to batch normalization, a Tensorflow inherent function that reduces value range variability. This is another transformation suggested by similar case studies and once again ended up improving the model’s performance quite noticeably.
- The data, read as a Tensorflow variable, would form the input layer of our neural network, otherwise known as the beginning of the training process.
- Immediately after the input layer, 3 convolution steps followed. Each of those had its own convolution, non-linear activation function (in our case ReLU) and pooling layer (in our case max-pool).
- Following the 3 identical layers, a fully connected layer was placed where the MLP structure was applied. This produced the output (a soft-max prediction for each image).

Apart from this solution, we experimented quite extensively with the parameters within each layer (e.g. filter size, number of hidden layers, etc.). Due to the fact that the MLP algorithm produced inferior results (only went up to 86% accuracy), we focused our efforts towards the CNN algorithm. What we consider deliverable solutions (albeit not optimal) are presented in the table below for easier comparison between results.

Normalization type	Gray-Scaling	Number of hidden layers	Filter(moving window) Size	Depth (layer 1, layer2, layer3)	Number of Iterations	Keep-Probability	Highest Accuracy Received
Batch Normalization(β = 0.0, γ = 1.0, ε = 0.001)	NO	128	5x5	(16, 32, 64)	50000	0.9	94.22%
Batch Normalization(β = 0.0, γ = 1.0, ε = 0.001)	NO	256	5x5	(16, 32, 64)	50000	0.9	94.09%
Batch Normalization(β = 0.0, γ = 1.0, ε = 0.001)	NO	128	5x5	(16, 32, 64)	50000	0.8	94.49%
None	NO	128	5x5	(16, 32, 64)	50000	0.8	91.21%
Batch Normalization(β = 0.0, γ = 1.0, ε = 0.001)	NO	128	5x5	(8, 16, 32)	50000	0.8	93.15%
Batch Normalization(β = 0.0, γ = 1.0, ε = 0.001)	NO	128	5x5	(16, 32, 64)	30000	0.8	93.01%
None	YES	128	5x5	(16, 32, 64)	50000	0.8	90.93%
Batch Normalization(β = 0.0, γ = 1.0, ε = 0.001)	YES	128	5x5	(16, 32, 64)	50000	0.8	94.29%
Batch Normalization(β = 0.0, γ = 1.0, ε = 0.001)	NO	128	3x3	(16, 32, 64)	50000	0.8	94.02%
None	NO	128	5x5	(32, 64, 128)	50000	0.8	20.05%

For all implementations, batch size remained 50. Only the number of iterations varied.

It should be noted that human accuracy for this particular dataset was 98% (this can be considered as the optimal performance an algorithm could hope to achieve and is also set as a goal by Google). Similar case studies produced accuracies quite close to ours. Those are available in the following link:

http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#5356484e

Taking these factors into consideration, we consider our implementation to be quite successful.

Conclusion

After experimenting with several different methodologies, algorithms and parameters we can summarize our findings to a few conclusions that could potentially assist future researchers. Those are the following:

- Adding the extra data (containing easy cases) to the training set helps with accuracy quite considerably.
- CNN outperforms MLP (as expected since it specializes in image processing).
- Gray scaling reduces accuracy quite a lot (about 10% for MLP and 5% for CNN).
- Despite the presence of noise or multiple digits per image, the algorithm seems to be able to distinguish the central digit successfully.
- GPU increases speed of execution a lot compared to a traditional CPU. If a compatible GPU is available it should almost always be used.

This study has produced a result comparable with published implementations of the SVHN dataset. However, due to our lack of knowledge of complex algorithms, room for improvement certainly exists. Given the nature of neural networks, there could potentially be a parameter combination (such as adding more layers or changing image dimensions) that could produce significantly better results. Unfortunately, this is rarely something the researcher can know beforehand and requires several attempts to be achieved.

Therefore, our suggestion to future researchers is to experiment with parameters a bit more, possibly adding more hidden layers to create a complex (and potentially more accurate) network.

References

- A Quick introduction to neural networks: https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/
- An Intuitive Explanation of Convolutional Neural Networks: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/
- Batch-normalized Maxout Network in Network; Jia-Ren Chang, Yong-Sheng Chen
- Convolutional Neural Networks Applied to House Numbers Digit Classification; Pierre Sermanet, Soumith Chintala and Yann LeCun

Appendix I: Team Roles

For the purposes of the assignment, a great deal of research needed to be conducted. All the team members contributed equally when it came to that, finding similar cases, popular methodologies and code fragments to face any difficulties or questions that occurred.

Furthermore, any decisions regarding implementation or suggestions of ideas were made during the numerous group calls and conversations, once again by all members without discrimination. As such it is difficult to divide the work done from this aspect, since every member participated in every part of the project.

That being said, each team member ended up focusing more towards what they were more proficient at or what the circumstances demanded. More specifically:

- Stamatis, being the only one who managed to set up GPU support for Tensorflow, was the one who focused more on implementing different algorithms and switching around the model's parameters.
- Giannis, having previous experience with MATLAB and image processing, was the person who led the preprocessing of the data and produced any required charts and statistics.
- Lefteris focused more towards producing a report and presentation, explaining the methods used and presenting the team's results.