Capstone Project

David Chainho

Machine Learning Engineer Nanodegree

October 28th, 2016

I. Definition

Project Overview

The project's objective is to build a machine learning model that can classify number strings from real-world images.

The problem is in the field of Computer Vision, although for this model, no Computer Vision specific algorithms are used to resolve the digits' localization, segmentation or classification problems.

In order to develop the model, it was used a dataset composed by house numbers obtained from real world images in Google Street View (SVHN). This dataset is very used in machine learning and object recognition fields because it has minimal requirements on data preprocessing and formatting and is representative of real world data.

Character recognition is a problem solved for many years, studied at first in the context of scanned documents and books [8, 9]. Recognizing handwritten digits as also been addressed [10]. Solutions from some years ago solved the problem of handwritten digits using a Linear Classifier, Nearest Neighbor Classifier and Neural Networks, LeNet 1 and LeNet 4 to name a few [11]. Only a few years ago has this problem started to being resolved with Deep Convolutional Neural Networks with higher success, mostly due to increased computer power [12].

One application of interest for solving this problem, is identifying house numbers in fronts of buildings. This would allow more accurate maps for navigation systems.

Problem Statement

This machine learning problem is one of object and text, in this case, digit recognition in a real world image. The objective is to develop a model that recognizes a sequence of

digits in a real-world image. The images in the data set have a sequence of up to 5 digits, from 0 to 9.

Hence, this machine learning problem is one of classification (prediction of discrete classes, the digits from 0 to 9). This problem will be solved using a supervised learning model, where the input of the model is a training dataset composed of real-world images, which contain a sequence of digits in natural scenes and corresponding labels, the digit sequence. The model will be tested against a similar dataset (training dataset), where the output of the model is the sequence of digits it detects in each image.

The strategy to solve this problem is to develop and train a Deep Convolutional Neural Network, that operates directly in the image pixels, without computing Computer Vision features. The model used to resolve the multiple digit learning is based in Goodfellow *et al.* (2013) [1] work. This solution can identify up to 5 digits in the same image.

It is expected for the model to recognize digits and identify them and the corresponding sequence.

Metrics

The metric to measure the performance will be the Softmax score function, in the validation and test set. Therefore, the prediction is:

- $s = s_1, \dots, s_n$: prediction for each digit (sequence of digits);
- X: input image;
- $S = S_1, \dots, S_n$: output sequence;
- Prediction function is:

$$s = argmax\ P(S|X)$$

The most likely prediction of the prediction vector (argmax) is obtained and compared with the real label. The accuracy of the model is the ratio of correct predictions:

$$accuracy = \frac{correct\ predictions}{dataset\ size}$$

It must be taken into account that the number of correct predictions for each sample is up to five, i.e., the number of digits being predicted. Therefore, the dataset size is also multiplied by the sequence length of five.

Accuracy was the chosen metric, mostly because is the most common used metric in similar works [4] in this dataset. So this metric allowed a direct comparison to previous work.

The Accuracy metric has the problem of bias towards the most common class, but as it can be seen in the next chapter, this metric can be applicable to this problem. There isn't one class that is much more dominant than others. The major biases that exist in this dataset are:

- Almost 90% of images contain a digit sequence of 2 or 3 digits, being 3 digits the most common;
- For S_1 digit position, digit '1' is present in almost 30% of the samples;

But given the very high number of digit combinations, given it's 11 labels (digits 0-9 and no digit) for 5 positions, the most occurring combination is still a very low percentage of the overall dataset.

II. Analysis

Data Exploration

The datasets are composed by images with multiple digits, obtained from real world Google Street View images. The images are from building numbers.

The input space varies in size, having the following ranges:

- Smallest Image size: 300 pixels
- Biggest Image size: 438876 pixels

For each image, a list of bounding boxes corresponding to each digit is also provided.

In this project, images with more than 5 digits are discarded.

Figure 1 Images from SVHN dataset and corresponding bounding boxes for each digit

The original datasets are composed by:

- 10 classes, one for each digit. For all digits, the class corresponds to its real value. For example, digit '1' has Class 1.
- The number of samples for each dataset is:
    - Training dataset: 33402 images
    - Testing dataset: 13068 images
    - Extra dataset: 202353 digit images, less complex, that can be used for extra training

Images vary in size and shape. Below is a table that shows the minimum and maximum sizes of the images of the various datasets:

	Minimum	**Maximum**
Width	22	1083
Height	12	516
Pixel Count	300	478686

Table 1 Minimum and maximum values for Width, Height and Total Pixel Count of all datasets

Since the images vary in size, and therefore, the digits' size in each image, a normalization procedure is executed.

In order to have a standard size for the input images, to work with the Convolutional Network, it was obtained a cropped image of size 32x32. The crop was done on digits' part of the images, using the bounding box info.

Another normalization procedure done was to use all the images in gray scale, in order to reduce complexity of the images. For the purpose of digit recognition, the color information should not matter in recognizing digits, only the form of the digit itself. Hence all images were converted from RGB color space, to YUV color space, and only

the Y channel was used for the images, which is the Luminance component, which should have enough contrast for digit recognition.

Exploratory Visualization

The provided dataset is a set of images and corresponding bounding boxes for each digit present in the image. The following analysis are for the training and extra datasets.

The distribution for each digit per S_L is the following:

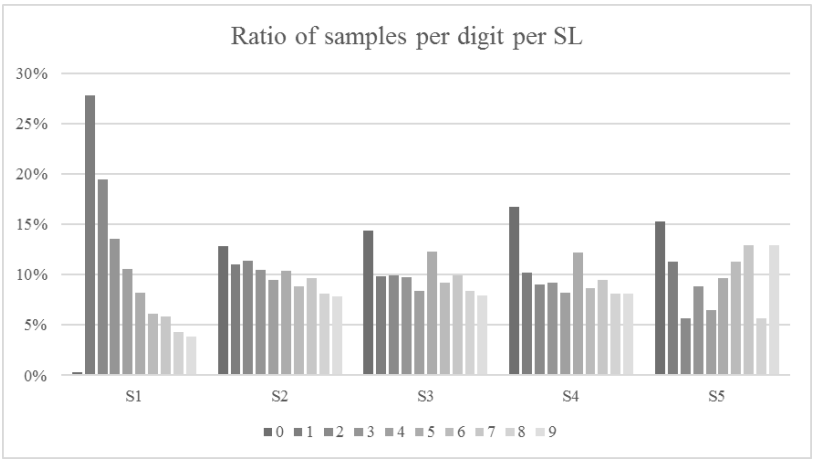

Figure 2 Ratio of digit occurrence per sequence position. It can be seen that for S1, digit '0' has very few occurrences, making S1 really unbalanced

For S_1 the distribution is unbalanced, with much more occurrences for digit '1' and very few for digit '0'. For the other digit sequence, the distribution is more balanced.

The total number of occurrences for each digit is the following:

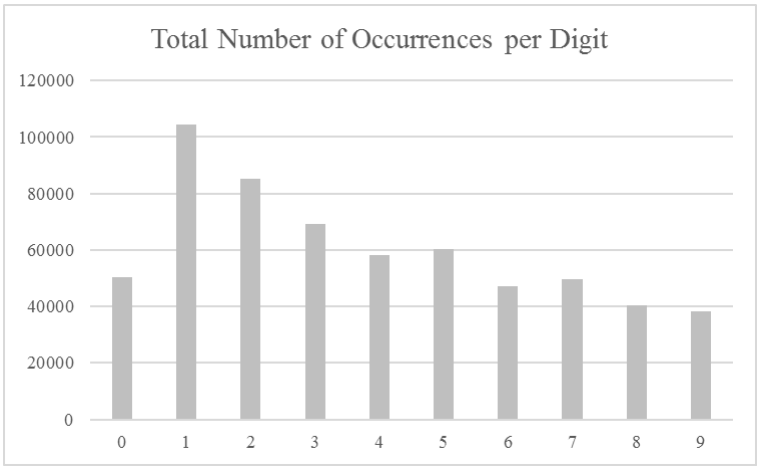

Figure 3 In this graph it's plotted the number of occurrences for each digit, for all images and sequence position

The datasets can be said to be slightly unbalanced, with a skew for digits ‘1’ and ‘2’.

Below can be depicted the total number of images per length of digits’ sequence. It can be seen that the majority of house numbers in this dataset have a length of 2 or 3 digits.

Figure 4 Histogram of number of images per length of digits sequence present in each image

Algorithms and Techniques

The applied learning model is a Deep Convolutional Neural Network, based in Goodfellow *et al.* (2013) [1] architecture, which yielded good results.

Convolutional Neural Networks (CNNs) are similar to regular Neural Networks, in the sense that it takes a set of inputs, performs a dot product and follows with a non-linearity. The major difference is that CNNs assume the inputs are images which results in optimizations of the network architecture and in less parameters.

In CNNs, neurons are arranged in 3 dimensions, width, height and depth. Unlike a regular Neural Network, where neurons are in a fully-connected manner, in CNNs, neurons in a layer are connected only to a small region of the prior layer. In CNN’s, a layer’s parameters consist in a set of learnable filters. These filters are small, spatially, applying only to a small part of the input. Then, each filter is passed along the whole width and height of the input volume, outputting an activation map. These activation maps are responses of the filter at every spatial position, thus reflecting visual features, such as edges or patterns. The activation maps are stacked along the depth dimension, resulting the output volume.

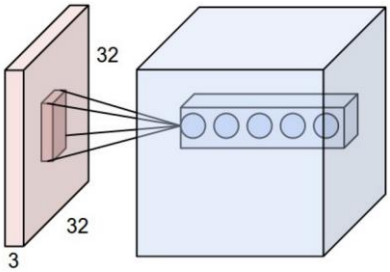

Figure 5 Example of a convolutional layer (in blue), connected to a input layer (in red) [13]

CNNs models usually have three main types of layers: Convolutional Layer (described above), Pooling Layer and Fully-Connected Layer:

- **Pooling Layer**: commonly inserted between consecutive Convolutional layers, to reduce spatial size, and as a result, reducing amount of parameters and computation in the network. This allows also to control overfitting. Most common form is pool layers with filter of 2x2, reducing height and width of its input, in half;
- **Fully-Connected Layer**: like a regular Neural Network, neurons are fully connected to all activations in the previous layer. Usually placed in the end of the network.

The CNN of this work has the following characteristics.

- Convolutional Layers:
    - 6 Convolutional Layers;
    - Each Convolutional Layer was followed by a ReLU Unit;
    - Convolutional Layer 2 and 4 are followed by Local Response Normalization and a Max Pooling Layer with 2x2 Subsampling Layer;
    - Dropout Layer applied at the last Convolutional Layer, to reduce overfitting.
- Fully Connected Layer:
    - Final Network Layer, with 5 Logits, one for each digit that can be present in the image

To reduce overfitting, it's also commonly applied Dropout layers. Dropout layers [14] are a simple way to reduce overfitting of neural networks, by dropping out some of the activations of a specific layer. The end result of applying dropout is similar to having an

ensemble of neural networks, i.e., an average of results from multiple neural networks, with a high computing efficiency.

This prject's CNN diagram can be depicted below:

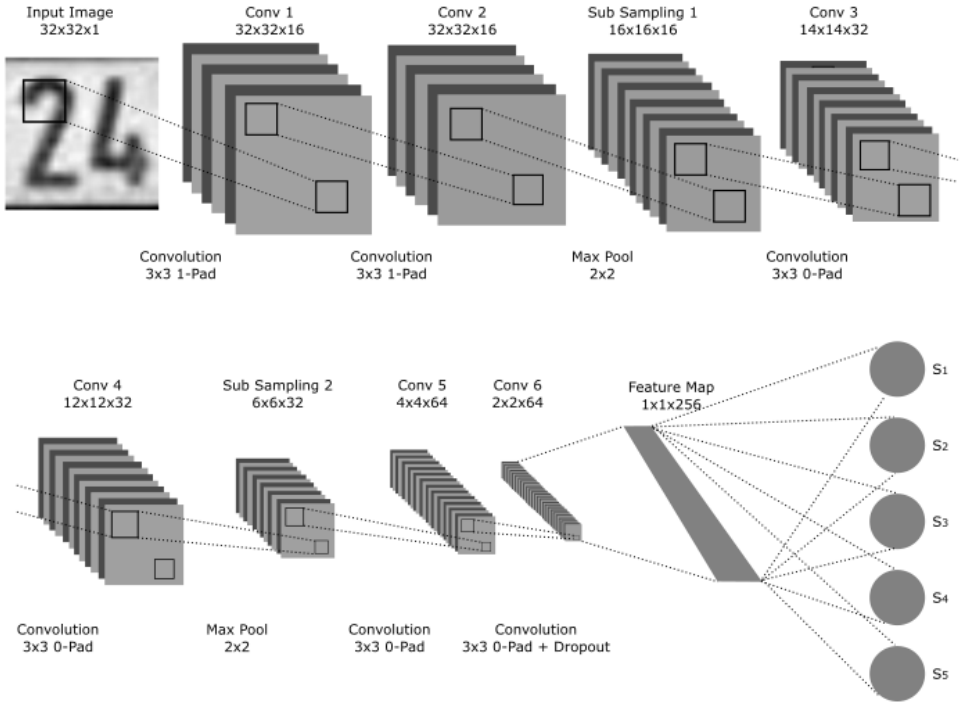

Figure 6 Deep Convolutional Neural Network topology

There are a set of parameters and network configurations that can be optimized to improve the accuracy of the model. These are:

- Training Parameters:
    - Training Length (number of epochs)
    - Learning rate (initial) and decay
    - Type of Optimizer (Adagrad, Adam, etc.)
    - Dropout rate
- Deep Convolutional Neural Network Topology:
    - Number of convolutional layers, or, depth of the network
    - Depth of convolutions, for each layer, i.e., number of filters
    - Learning rate optimizer algorithm (used in this project is Adagrad)

This model takes input data in batches, to optimize memory usage, for the training phase. A LeCun Local Contrast Normalization ([6]) is applied at the input of the Convolutional Layer, using the solution provided in [7].

Benchmark

Accuracy is calculated by getting the Softmax score with highest probability and comparing with real value. Then it's just a ratio of correct answers over all possibilities.

The benchmark to beat is 96.03% accuracy as presented in Goodfellow *et al.* (2013) [1], which is the model implemented in this capstone project. The benchmark has a few differences from this project, being the main ones:

- Cropped images size is 64x64, followed by 54x54 crop within the first crop, shifted randomly to provide several versions of the same image. This provides a data augmentation method, not used in this project
- A deeper network, with 8 convolutional layers (vs 6 in this project). Each of these layers have a higher depth, i.e., number of filters (48, 64, 128, 160 and 192 vs 16, 32 and 64 of this project)
- Dropout applied to all layers. In this project it's only applied to the last convolutional layer

These differences are mainly to limited computer resources available in the system used. Therefore, the objective of this project is to verify how close to the benchmark value it's possible to get with a less deep, thus, less computationally demanding network.

III. Methodology

Data Preprocessing

Due to the nature of this problem (digit recognition), some algorithms were applied in order to normalize the dataset and, therefore, help to reduce variability and dimensionality of the problem.

- Images are cropped around the digits bounding boxes, with an extra 15% of width and height around the digits;
- Images are reduced in depth, from RGB to Gray scale. Gray scale is obtained by converting from RGB to YUV, and using the Y channel, the channel with the most relevant information (light intensity giving contrasts);

- Resize to standard size of 32x32;
- For each image, it’s performed mean subtraction and normalization, by dividing by its standard deviation.

One image with more than 5 digits was removed from the dataset, as it was considered as an outlier.

An example of post processing can be seen below:

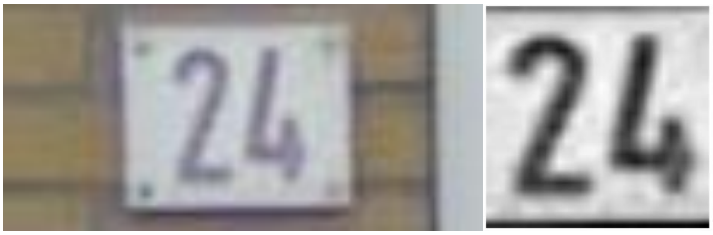

Figure 7 Left: Original image; Right: Preprocessed image

Implementation

The Deep Convolutional Neural Network was implemented with Tensorflow library.

The first stage of the implementation consisted in retrieving the images, cropping the images by the bounding boxes limit, create the training and validation dataset from training and extra image set, and pre-process each image.

All the created datasets were save in Pickle files to be loaded later for the training phase.

The training model was made like this:

1. Load all the datasets from the saved Pickle files
2. Create the CNN using Tensorflow library
3. Define CNN topology, parameters
4. Define the loss and accuracy functions
5. Training the network, logging all the batch and validation accuracy and training loss
6. Analyze the final test dataset accuracy and verify value
7. Tested different parameter values for Dropout and initial Learning Rate parameters

Refinement

In order to improve the final test dataset accuracy of the model, it was trained the network with the following initial parameters:

- Dropout rate (keep probability): 0.55
- Initial Learning Rate: 0.03
- Number of Epochs: 200

It was plotted the training and validation set accuracy over training steps and training loss.

Then, guidelines from course CS321n of Stanford University [2] were applied to results obtained, to improve the model. These consisted in:

- Analyzing the Loss plot to get feedback about the Learning Rate
- Analyzing the Training and Validation set accuracy plots to verify if overfitting exists

IV. Results

Model Evaluation and Validation

To validate the model, a validation dataset was created as follows:

- 600 samples for each digit of S_1 (first digit) was obtained from extra datasets, in an attempt to have a balanced validation dataset
    - The exception is for digit '0', because of the reduced number of samples that have digit '0' in S_1 position. For digit '0' it was extracted 100 samples from extra dataset
- The final size of the validation dataset is 5500 samples
- The training dataset used for model training was the original SVHN training set with 33401 samples
    - Extra dataset was not used for training due to memory restrictions of the system

The neural network was trained a batch size of 64 samples. This resulted in the following:

- 1 epoch: 522 iterations

- Training duration for parameter tuning: 200 epochs, or, 104400 iterations
- Average runtime per training trial: approximately 11 hours

The model was evaluated with a baseline network topology and with some default values for the following parameters, as described in Refinement chapter. The results were the following:

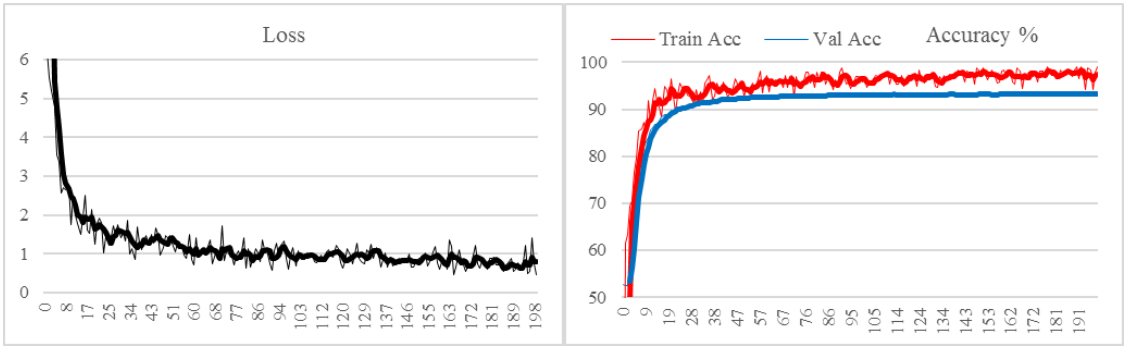

Figure 8 Baseline implementation. Left: Loss over training epochs; Right: Training and Validation dataset accuracy over training epochs

The result of the first run was quite satisfactory, giving a Test Set Accuracy of 94.10%.

The Loss plot seemed to indicate a slightly high Learning Rate, so it was made a second run with an initial Learning Rate of 0.003. This run yielded worst results, with a final Training Accuracy of 92. 2% and Validation Set Accuracy of 84.9%:

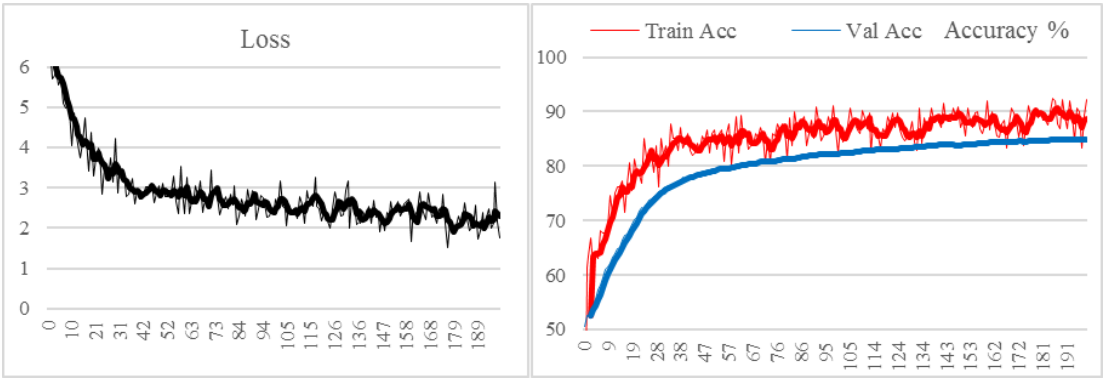

Figure 9 Training trial with initial Learning Rate of 0.003. Yielded worst results than Baseline implementation. Left: Loss over training epochs; Right: Training and Validation dataset accuracy over training epochs

The Baseline implementation default Network Topology and default parameters seem to yield very satisfying results, reaching a 94.10% Test Accuracy.

Justification

The final accuracy for the test dataset was 94.10%. This value is below the benchmark by 1.93% (96.03%), as expected, for the reasons explained in the Benchmark chapter. It's worth noting, however, the difference to the benchmark is relatively small, if it's taken into account the computational resources required by both models.

The final solution provides a good accuracy, although quite far from the state of the art for the reasons stated above.

The model with this level of accuracy can't be used to replace humans in labelling of real world images (which achieve 98% accuracy) although it can provide automation for some applications that can tolerate some error.

V. Conclusion

Free form Visualization

Below are some samples labelled by the model:

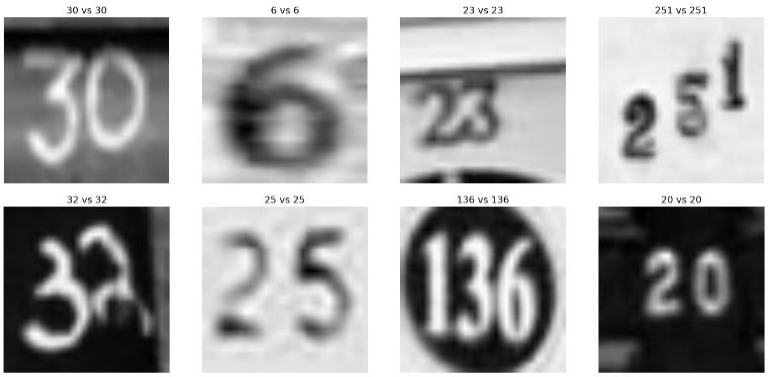

Figure 10 Some Test set samples labelled correctly by the model

The model seems to generalize well to different digit formats and contrasts. It can be seen that both digits with a dark tone and digits with a white tone are labelled correctly, demonstrating the model has learned the digits' forms and color has no impact.

For example, for the samples with digit sequence "32", the model labelled the digit '2', despite being, apparently, hand written, versus other forms of the same digit in the other images having a more similar form.

In the images below are some samples of testing dataset where the model fails.

Real: 8; Prediction: 1 **Real: 21; Prediction: 22** **Real: 5; Prediction: 1**

Figure 11 Some Testing set samples where the model fails to correctly label the house numbers

Reflection

The process for this project was the following:

1. Identify a relevant public and popular dataset to apply deep learning
2. Preprocess the dataset with relevant techniques
3. Identify adequate deep learning architecture to resolve the problem
4. Identify a good benchmark to try to beat
5. Decide on the library to develop the model, in this case, Tensorflow
6. Train the model several times, with different parameters and architectures

The most interesting aspect of the project was to learn about convolutional neural networks and their applicability.

The most difficult part was to apply the theory of CNNs to the Tensorflow library. Although many examples and tutorials exist, it took some time to intuitively understand how the library works. Then, due to the system used to develop and train the models having low specifications, memory constraints problems had to be dealt with.

This greatly affected the ability to test deeper networks, with more hyperparameters. Another great handicap was the training times to test different parameters, where to test models in 200 epochs, took close to 11 hours.

Improvement

The algorithm could have better improved by using a higher end system, with much more RAM memory and dedicated GPU with CUDA, to fully explore Tensorflow's capabilities.

Other techniques and algorithms that could have been attempted to use would be Inception CNNs, which have very promising results [3]. This networks consist of having multiple convolutional types in parallel which result in a feature vector to a Fully Connected layer.

Using this project's final results has the benchmark, other solutions exist that could provide better results [4].

VI. References

[1] Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S., & Shet, V. (2013). Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*.

[2] http://cs231n.github.io/neural-networks-3/#baby

[3] Liao, Zhibin, and Gustavo Carneiro. "Competitive Multi-scale Convolution." *arXiv preprint arXiv:1511.05635* (2015).

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

[5] 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.

[6] Jarrett, Kevin, Koray Kavukcuoglu, and Yann Lecun. "What is the best multi-stage architecture for object recognition?." *2009 IEEE 12th International Conference on Computer Vision*. IEEE, 2009.

[7] https://github.com/jostosh/theano_utils/blob/master/lcn.py

[8] G. Nagy. At the frontiers of OCR. Proceedings of IEEE, 80(7):1093–1100, July 1992.
[9] S. Mori, C. Y. Suen, and K. Yamamoto. Historical review of OCR research and development. Proceedings of IEEE, 80(7):1029–1058, July 1992.

[10] R. Plamondon and S. N. Srihari. On-line and off-line handwriting recognition: A comprehensive survey. IEEE Trans. Pattern Anal. Mach. Intell, 22(1):63–84, 2000.

[11] Bottou, Léon, et al. "Comparison of classifier methods: a case study in handwritten digit recognition." International conference on pattern recognition. *IEEE Computer Society Press*, 1994.

[12] Sermanet, Pierre, Soumith Chintala, and Yann LeCun. "Convolutional neural networks applied to house numbers digit classification." *Pattern Recognition (ICPR), 2012 21st International Conference on. IEEE*, 2012.

[13] http://cs231n.github.io/convolutional-networks/

[14] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research 15.1* (2014): 1929-1958.