C-NN & Street View House Numbers Dataset

A Convolutional Neural Network model to predict multi-digits over the SVHN dataset

Technological University Dublin, Ireland
Lecturer: Robert Ross

Bruno Ambrozio
Student ID: D16128063
Class: DT228B/DA
bruno.ambrozio@mydit.ie

Victor Zacchi
Student ID: D16128783
Class: DT228B/ASD
victor.zacchi@mydit.ie

***Abstract* — This work demonstrates the process of single and multi-digit classification by using Convolutional Neural Network (CNN) over the Street View House Number (SVHN) dataset. It was built up upon the paper "*Decoding a Sequences of Digits in Real World Photos Using a Convolutional Neural Network*"[7]. The problem to be solved is a multi-digit classification. In this project, both datasets from the SVHN are used. The cropped images were used to produce a single digit classifier, while the full one for multi-digit classification, which in turn makes use of the first one by transfer learning workflow in order to build up the final model.**

Keywords— Deep Learning, CNN, SVHN, TensorFlow, image classification

I. Introduction

One of the main goals in artificial intelligence (AI) is to make the machine learn from experiences (by analysing images, sounds and data in general) of a given problem, similar how human brains work. Deep Learning make use of techniques for learning levels of representation and abstraction extracted from real-world examples to understand "things" close to their actual meaning. The performance of the machine learning methods generally depends on the choice of the representation of the data, which they are applied, since the extraction of characteristics consists in associating a vector of characteristics for each image so that the images of the same class are represented by similar vectors, that is, next in the feature space.

In this paper we outline 2 C-NN model implementations to classify single and multi-digit numbers from images. They were built up upon the paper "Decoding a Sequences of Digits in Real World Photos Using a Convolutional Neural Network"[7] and fully trained over the SVHN datasets.

This paper analyses Diesel's implementation [7], adapt and improve the code, and provides an API that makes possible to use the final classifier against single images.

The paper goes over the data preprocessing strategy adopted, the model implementation, briefly explains the CNN layers in use and finalises with the results and considerations for further improvements.

II. Dataset and Tasks

A. Dataset

The SVHN dataset [10] was derived from house numbers of images in Google Street View. The database contains more than 600 digital images, of which 73,257 are images for training, 26,032 images for tests, and 531,131 additional pages, simplified, that can be used as a complement to a training set. The characters in this dataset contain only numbers. The SVHN is available in two formats:

Original images have variable resolution and with character level bounding boxes.

Cropped, where the bounding boxes of the characters are extended into the appropriate dimension to become square windows, so that they are resized to a fixed size, 32-by-32 pixels. This resizing does not introduce aspect ratio distortions, although some images have parts of other adjacent digits visible. Both dataset formats are used in this project (excluding the extra images) which is capable of predicting a single-character or multiple characters in real-time.

B. Tasks

With Convolutional Neural Network we have trained a Tensorflow model capable of identifying digit sequences on images using the SVHN datasets. The tasks involved to produce the results were the following:

1. Firstly, our program downloads the dataset of single well-cropped house numbers, as well as the irregular sized images.
2. Separating them out into different folders from training and test data.
3. Using the DigitStruct.mat to identify the correct location of the digits in the full-sized images, we crop and resize each into a 64x64 regular dimensions.
4. Next, the model is trained to classify and recognize a single digit.

5. The weights are from the single digit classifier is saved to be used in the training of the multi-digit classifier, by transfer learning workflow [6].
6. Finally, train our model to classify up to 5 digits reusing the weights from previous steps.

III. Model Design

A. Data Preprocessing

The SVHN dataset for the train and test files, used in this project, contains approximately 100,000 images in a where 60 percent is used for training, 30 for testing, and a further 10 percent split from the training dataset which will be used for validation.

For the classification of a single digit, we have used the already cropped dataset. These images have already been processed and converted into .mat files containing two variables, ***X*** which is a 4-D matrix containing the images, and ***y*** which is a vector containing the class labels for each of the individual digits.

For the next phase, the multi-digit recognition, the full unprocessed images were needed. Both the training and test data sets contained a bounding box reference file (digitStruct.mat) which had the necessary information to find out each of the digits in our images. Such as a *Filename* used to point to the correct image file; the *Bounding Box* the exact pixel coordinates (top, left, height and width) for each of the digits in a given image; and the *Label* of each digit.

This process is a lot more complicated due to the fact that we have to first preprocess the image by iterating through the struct file to crop and resize our images accordingly.

The image below represent how the resizing and cropping was done. Firstly, the measuring of bounding boxes was taken from digitStruct file to locate the digits in the picture; secondly, an extra 20% padding is calculated to ensure all numbers were fully visible; and finally, the image is cropped to 64 by 64 closely to where the numbers are located.

figure 1 - bounding boxes and cropping

The dataset contained some images with more than 5 digits, for simplicity only 5 digits images were kept in the dataset during the preprocessing phase.

One-hot-encoding - A tricky part is the creation of the one hot encodes for each of the labels was that this dataset contains the digit “0” denoted as 10 as the label which can be misleading. The labels had to be adjusted where 10 is converted to 0 and the 10 value was used for the notation of a null digit. Diesel’s experiment [7] also made use of another interesting technique which was to append a number prefix identifying the length of the digit sequence. For example, the target number of the image 1 “995” array would look like this [‘3’, ‘9’, ‘9’, ‘5’, ‘10’, ‘10’] where the first digit indicates the total digits in the sequence followed by the other three elements consisting of the correct numbers, and the 10’s representing empty values. Other tutorials, however, such as Almenningen [2] and another from Chuang [4] does not seem to have made use of such prefix, instead, it reads up until it finds a 10 digit.

B. Convolutional Neural Network Layers

The model was developed using multiple layers of convolutions. Each layer has a simple function which is used to transform a 3D volume of input with a differentiable function, as per outlined by Diesel [7]. Each individual layer of the CNN implementation consists of a copy of the same neuron where each neuron has their set of weights and biases parameters. All images are passed through the network multiple times, by many epochs, although the weights correction is done through back-propagating of the error in smaller samples (batches) of the data. The network calculates the gradient and adjusts the weights at the end of each mini batch.

The CNN model developed is composed of the following component layers.

Convolution Layer: Convolution is the process of calculating the intensity of a determined pixel in the function of the intensity of its neighbors. The convolution layer performs two-dimensional filtering on the image parameters calculating the product of the entries of the filter and the input at any given position. Such filters are used to produce features maps from different layers of the images.

Max Pooling Layer: The output of the max-pooling layer is given by the maximum activation of rectangular regions without overlap. This layer reduces the resolution of the features map as well as reduce the sensitivity from displacements and other forms of distortion in the image. Thus, this technique can help reduce network computation and in avoiding overfitting of our model.

Activation function: basically decide whether a neuron should be activated or not. That is, whether the information the neuron is receiving is relevant to the information provided or should be ignored. The activation function used is Rectified Linear Unit (ReLU), the main advantage of using this function over other activation functions is that it does not activate all neurons at the same time. Therefore, if the input is negative, it will be converted to zero and the neuron will not be activated. This means that at the same time, only a few neurons are activated, making the network sparse, efficient and easy for computing.

Dropouts: In its simplest form of the dropout algorithm, during the training, each insertion of a new data vector in the network, occurs the temporary elimination of neurons and their connections. Neurons remaining after elimination are trained through backpropagation. This technique reduces complex co-adaptations of neurons since a neuron cannot

rely on the presence of other neurons in particular. It is therefore forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.

Fully Connected Layer: this layer is used at the end of the network. This layer flattens out the feature maps from previous pooling layers.

Output Layer: this layer receives the flattened values from previous layers and uses a softmax activation function to best calculate the output probability for each of the digits in the output image.

Both the multi-digit and single-digit predictions were done using similar approaches as of above. The full architecture can be represented by the figure 2 below.

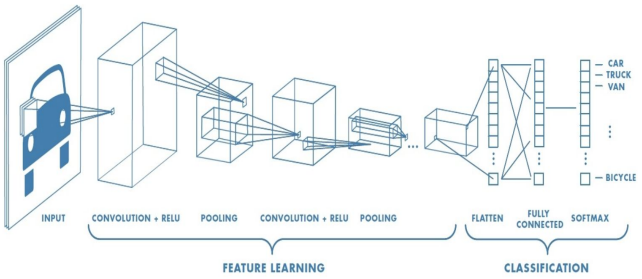


figure 2 - cnn architecture [6]

C. *The Model*

As previously mentioned, Diesel's CNN model [7], was divided into two main tasks. The first was to implement a transcription of a single digit using the cropped images from the SVHN dataset. After a satisfactory result from this process was achieved, the author moved onto the next phase re-applying the insights gained from the previous step to a more arduous large image and multi-digit processing.

The architecture of the smaller cropped images consisted of three convolutional layers with one locally connected layer. Each of the tensorflow ConvNet layers was set to use 5 by 5 filters and padding set to 'Valid' - which keeps the original dimensions of the convolutional input. Once the input is reduced to 1x1 dimension, the layer is flattened into a fully connected layer. The full sized images required one more convolution before flattening. Each of the layers is followed by a max-pooling layer. The pooling layer is used to reduce the spatial dimensions of the images a flat one by one feature vector map is achieved. Once flattened the vector is passed to a softmax classifier to obtain the class scores. For the multi-digit classification, a softmax classifier is attached to each digit in the sequence producing the correct class.

IV. Results / Conclusion

The model was trained in a MacBook Pro, Mac OS X Darwin Kernel (10.14.5-18F132) equipped with a CPU Intel Core i7 2.9 GHz of 4 cores (L2 Cache (per Core): 256 KB and L3 Cache: 8 MB) and 16GB of memory RAM. (Hyper-Threading Technology enabled).

After the preprocessing phase, where the datasets were downloaded and parsed as per outlined on part III of this paper, the train of the classifier model started (single digit) and resulted in the following metrics:

- Time taken for full train: 2h42mi48s
- Epochs: 128
- steps: 32900
- Final loss rate: 0.06
- Final learning rate = 0.000106
- 9.1 examples/sec
- 28.15 sec/batch
- Mini-Batch Accuracy: 1.00%
- Final test accuracy: 0.88281%

The regressor model (multi-digits classifier) resulted in the following metrics:

- Time taken for full train: 7h08mi32s
- Epochs: 128
- steps: 120200
- Final loss rate: 0.01
- Final learning rate = 0.00
- 1.5 examples/sec
- 20.742 sec/batch
- Mini-Batch Accuracy: 1.00%
- Final test accuracy: 0.91250%

It's important to restate that, the second model (multi-digit) start training by loading the checkpoints of the first one, by using the transfer learning workflow technique [6]. Therefore, the final accuracy of the whole model of this work is 91% after 256 epochs.

By analysing the final logs generated by the TensorFlow using the TensorBoard, it's possible to visualise that train accuracy rate gets quite stable after 10k steps, after 48mi15s training, and the validation achieves its peak after 17k steps, after 1h 21 min 15s training (see figure 3).

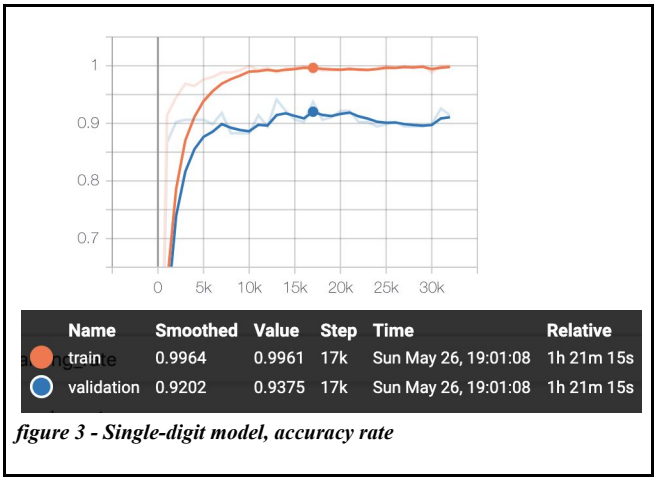
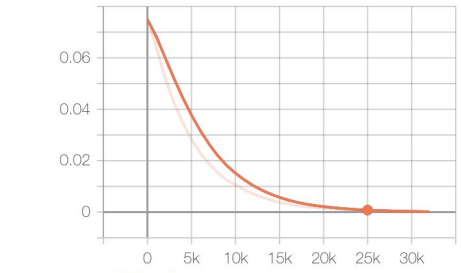


figure 3 - Single-digit model, accuracy rate

Single-digit model learn rate in turn, does achieve a stable value close to 0 after 25k steps, right after 2h 2mi 23s training (see figure 4).

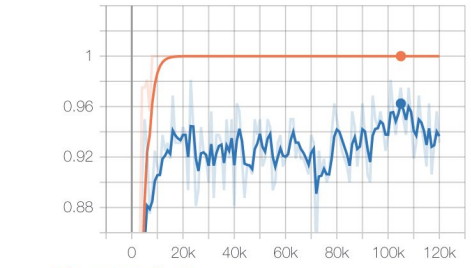


	Name	Smoothed	Value	Step	Time	Relative
●	train	7.8146e-4	5.1791e-4	25k	Sun May 26, 19:42:16	2h 2m 23s
●	validation	7.8146e-4	5.1791e-4	25k	Sun May 26, 19:42:16	2h 2m 23s

figure 4 - Single-digit model, learn rate

The multi-digit model though had a quite particular characteristic, although the train accuracy rate gets its peak (and kept it until the end) after the 21k steps, after only 1h 17mi train, the validation accuracy rate variated a lot, and achieved its peak on step 105k, after 6h 15min 51s (see figure 5).

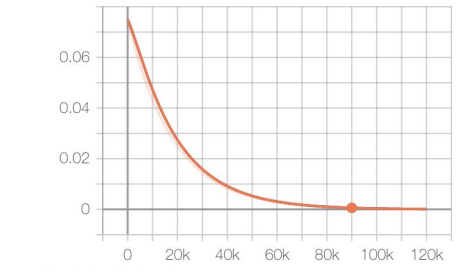
The point that may justify such non-smooth graph is due the multi-digits been trained with only 32 batch size, in contrast to the single-digit which made use of 256 batch size.



	Name	Smoothed	Value	Step	Time	Relative
●	train	1	1	105k	Mon May 27, 02:38:57	6h 15m 51s
●	validation	0.9624	0.975	105k	Mon May 27, 02:38:57	6h 15m 51s

figure 5 - Multi-digit model, accuracy rate

The multi-digit learn rate reached zero at ~90k step, after 5h 23min 34s (see figure 6).



	Name	Smoothed	Value	Step	Time	Relative
●	train	5.9542e-4	5.4519e-4	90k	Mon May 27, 01:46:39	5h 23m 34s
●	validation	5.9545e-4	5.4522e-4	90k	Mon May 27, 01:46:39	5h 23m 34s

figure 6 - Multi-digit model, learn rate

Given the results analysed on the graphs, perhaps a good fit would be implementing an early stopping technique. Although we already have a dropout to avoid overfitting, the early-stop could help saving hardware resource and training time, as the peaks were achieved way before the model completed running over all epochs.

Some manual tests were also performed by using either some images from the full dataset (scope test) and downloaded pictures from the internet.

figure 7 - Sample images for manual tests

An interesting scenario was observed with the manual tests. As per figure 7, we have the following number image labels: “123”, “60” and “60”, respectively. The “123” image is a jpg from the internet, which easily gets predicted by the model (it’s worth mentioning that in a previous train, the 3 was predicted as 8). Both images with a “60” are actually the same, however, the first is the result of the preprocessing phase of the model and the second one is the image as it came from the dataset. For the preprocessed image the model could predict correctly, but the second one, even if we manually cropped close to the numbers, the model was not able to find the correct prediction.

Although the high accuracy reported in both single and multi-digits training, our model did not perform well on other real-world images. We could notice that, although the images used for validation and test of our model were not part of the training process, its dimensions are still nicely adjusted as per the other samples from the training dataset, meaning that the model doesn’t generalize well or suffered from overfitting as it requires considerable adjustment to a new input image for a successful prediction.

For future exploration, some points that perhaps worth to explore are:

- Don’t use the image dimensions provided by the dataset specification in the validation dataset, in a attempt to improve generalization. (It will probably drive the accuracy in training down).
- Apply image augmentation and randomly change the image orientation, for better predictions on vertically aligned digits.
- Change the model to make use of R-CNN layer, in attempt to address better the spatial locations of digits for the multi-digit model.

V. References

[1] F. Adelkhani, Understanding Convolutional Neural Networks + Edge Detectors, ConvNets, and DIGITS. 2019 [Online]. Available: https://www.youtube.com/watch?v=SbLJLT8xXD0. [Accessed: 27-May- 2019]

[2] T. Almenningen, "05-svhn-multi-preprocessing.ipynb", GitHub, 2017. [Online]. Available: https://github.com/thomalm/svhn-multi-digit/blob/master/05-svhn-multi-preprocessing.ipynb. [Accessed: 27- May- 2019]

[3] T. Almenningen, "06-svhn-multi-model.ipynb", GitHub, 2017. [Online]. Available: https://github.com/thomalm/svhn-multi-digit/blob/master/06-svhn-multi-model.ipynb. [Accessed: 27- May- 2019]

[4] K. Chuang, "svhn-preprocessing.ipynb", GitHub, 2018. [Online]. Available: https://github.com/k-chuang/tf-svhn/blob/master/svhn-preprocessing.ipynb. [Accessed: 27- May- 2019]

[5] K. Chuang, "svhn-model.ipynb", GitHub, 2018. [Online]. Available: https://github.com/k-chuang/tf-svhn/blob/master/svhn-model.ipynb. [Accessed: 27- May- 2019]

[6] "Convolutional Neural Network", Mathworks.com, 2019. [Online]. Available: https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html. [Accessed: 27- May- 2019]

[7] B. Diesel, Decoding a Sequences of Digits in Real World Photos Using a Convolutional Neural Network. 2016 [Online]. Available: https://github.com/bdiesel/tensorflow-svhn/blob/master/capstone%20report%20final.pdf. [Accessed: 27- May- 2019]

[8] M. Levoy, K. Dektar and A. Adams, "Spatial convolution", Graphics.stanford.edu, 2012. [Online]. Available: https://graphics.stanford.edu/courses/cs178/applets/convolution.html. [Accessed: 27- May- 2019]

[9] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu and A. Y. Ng, "Reading Digits in Natural Images with Unsupervised Feature Learning", Ufldl.stanford.edu, 2011. [Online]. Available: http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf. [Accessed: 27- May- 2019]

[10] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu and A. Y. Ng, "The Street View House Numbers (SVHN) Dataset", Ufldl.stanford.edu, 2011. [Online]. Available: http://ufldl.stanford.edu/housenumbers/. [Accessed: 27- May- 2019]