Capstone Project

Machine Learning Engineer Nanodegree

Camilo Gordillo

04 November 2016

Definition

Project Overview:

As stated by Goodfellow et al. [1]: “Recognizing arbitrary multi-character text in unconstrained natural photographs is a hard problem”. Typically, the individual characters would be segmented out of the original image and a post-classification model would be employed to recognize these individual characters. Deep convolutional networks, however, are capable of unifying this procedure while operating directly on the image pixels.

In this project, I have created a Python application capable of recognizing number strings from real-world images taken from a webcam. The application uses a Deep Convolutional Network trained using the SVHN[1] dataset and following the architecture and approach described by Goodfellow et al. [1].

Problem Statement:

The goal of the project is to develop an application capable of identifying sequence of numbers within real-world images without the need of segmenting single digits beforehand. I would like to avoid the use of sliding window approaches, while also keeping the network complexity low (due to computational power limitations). Because of this, I will implement a 2-stage recognition system as the one shown in Figure 1.

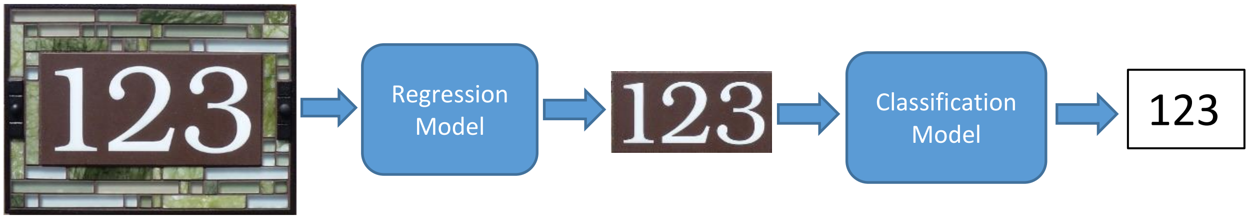

Figure 1 Implemented 2-stage recognition model

[1] The Street View House Numbers (SVHN) Dataset. *http://ufldl.stanford.edu/housenumbers/*

The job of the regression model will be to identify the location of the numbers within the image. It will output a set of coordinates corresponding to the bounding box containing the number string. I will crop this bounding box and I will pass it to a second stage (Classification Model), which will take care of recognizing which numbers can be seen in the image. I will train models which should be capable of recognizing number strings composed of up to 5 digits.

I will approach this problem by following these steps:

1. Generate a synthetic dataset by concatenating character images from the MNIST[2] database of handwritten digits.
2. Design and train a network architecture capable of achieving good performance on the synthetic dataset.
3. Create a more challenging and realistic scenario by downloading and preprocessing the SVHN dataset.
4. Train the model with this new dataset and evaluate its performance.
5. Design and train a regression model using the bounding boxes provided by the SVHN dataset.
6. Develop an interface to load video frames from a web camera and to identify number strings within those frames.

Metrics:

Like Goodfellow et al. [1], I will determine the *accuracy* of our Classification Model by computing the proportion of the testing images for which the length of the sequence and every element or digit in the sequence is predicted correctly. There will be no 'partial credit' for correctly classifying individual digits.

The performance of our Regression Model, however, will be measured in terms of the *Dice's coefficient* Q_S, which measures the similarity between the predicted Bounding Box (*X*) and the ground truth (*Y*). $|X \cap Y|$ represents the overlapping area between the predicted and the real bounding boxes. A coefficient of 1 represents a perfect match.

$$Q_s = \frac{2|X \cap Y|}{|X| + |Y|}$$

[2] The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/

I will also compute the *accuracy* with which the Regression Model is capable of identifying whether or not there is a number in the image.

Analysis

Data Exploration:

The Street View House Numbers (SVHN) dataset was obtained from house numbers in Google street view images. The dataset contains over 600000 digit images and represents a more real and challenging dataset when compared to the MNIST database.

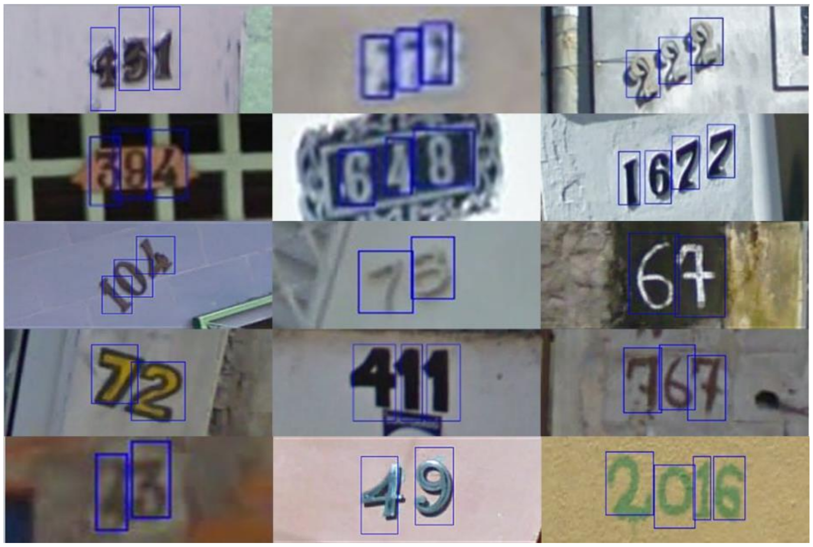

Figure 2 Images from the SVHN dataset. Taken from http://ufldl.stanford.edu/housenumbers

The bounding boxes in Figure 2 are shown just for illustration purposes given that this information is stored in an additional file. The downloaded folders contain both the original images in *png* format and a *digitStruct.mat* file. The *digitStruct.mat* file contains a structure called ***digitStruct*** which is organized as follows:

- digitStruct [struct]: structure with the same length as the number of images
  - name [string]: filename of the corresponding image
  - bbox [struct]: structure containing the bounding boxes and labels
    - label [int]: number in the current bounding box
    - height [int]: height of the bounding box
    - width [int]: width of the bounding box
    - left [int]: x position of the top left corner
    - top [int]: y position of the top left corner

Exploratory Visualization:

The SVHN dataset is composed not only by training and testing samples but also by an additional, somewhat less difficult, extra set which can be used as additional training. The number of images which is available on each dataset is shown below.

Folder	*Number of images*
Training	33402
Testing	13068
Extra	202353

Because the dataset is composed by variable-resolution images, it is interesting to get some statistics regarding the size of the available images:

	Training	*Testing*	*Extra*
Avg. Height	57,21	71,56	60,80
Max. Height	501	516	415
Min. Height	12	13	13
Avg. Width	128,28	172,58	100,38
Max. Width	876	1083	668
Min. Width	25	31	22

As it can be seen in the table above, there is a wide range of image sizes in the available datasets. This is good because it will provide us with a lot of resolution and scaling variability.

It is also interesting to analyze the length of the number strings which can be found in the available images. If all of them contained only one digit, it would not be possible to train a multi-digit classifier. Knowing the distribution of number string lengths can help us to determine if the dataset is useful for us or not. Figure 3 shows the percentage of the datasets which contains number strings with a given length.

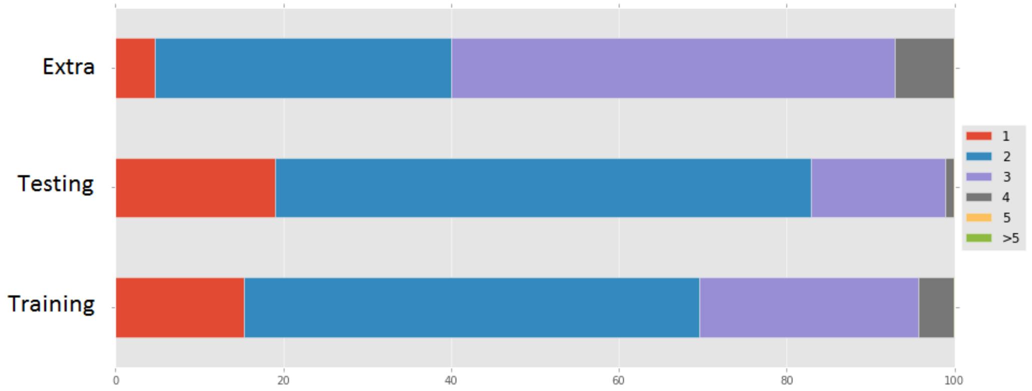

Figure 3 Distribution of the number string lengths in the three different datasets

From Figure 3 we can determine that the datasets are composed mainly by number strings with up to 4 digits. The number of images with 5 digits is negligible and by further analyzing the data I determined that there is only one image with more than 5 digits.

Based on these results, I will design the network to deal with up to 5 digits although I do not expect the model to perform well on images with this number of digits. It will be interesting to evaluate how does the model behave when dealing with 5-digit number strings giving the lack of training data.

Algorithms and Techniques:

I will try to replicate the classifier presented by Goodfellow et al. [1] which is based on a Convolutional Neural Network. Convolutional networks can represent very complex models and are currently the state-of-the-art technique for most image processing tasks. Moreover, this kind of architectures make the explicit assumption that the inputs are images and are very powerful and efficient when dealing with object detection problems.

There are several parameters which need to be tuned when optimizing the model performance. Although some of these parameters will be tuned with the help of a *Validation Set*, many others will be chosen based on state-of-the-art architectures, recent literature and computational constraints.

There are different ways of approaching the current recognition problem and a 2-stage solution as the one presented in Figure 1 is not strictly necessary. I could train a complex model capable of coping with scale and location variances, but that would require a lot of data, a lot of parameters and therefore, a high computational power and training time. A second approach could involve a sliding window technique, but I decided to go for something a bit more interesting and use a second convolutional network instead.

The first network (Regression Model) will take the original image (properly scaled) as input and it will output the following vector:

Index	Meaning
Output [0:1]	Probability of having a number string within the provided image (confidence score)
Output [2:5]	Bounding Box surrounding the number [Top, Left, Bottom, Right]

If a number is detected within the image, the system will crop the corresponding bounding box. This crop will be properly resized and passed to the second network (Classification Model), which in turn will output a classification probability for up to 5 digits. The Classification Model will consider 11 different classes: 0-9 to represent the corresponding digits, and 10 to represent empty spaces.

Benchmark:

Goodfellow et al. [1] achieved a sequence transcription accuracy of 96.03% on the SVHN dataset after training for about 6 days(!). My goal, however, will be to implement some more recent techniques to reduce the model complexity without affecting its performance too much. Given that I cannot compete with the kind of computational power which is usually employed when training deep models, I will focus on optimizing the network architecture. My ideal performance would be obtaining a transcription accuracy above 90%.

There is, however, not a clear benchmark for our Regression Model. It would be very good if we could test the recognition system using a webcam and obtain decent results.

Methodology

Data Preprocessing:

1. **GENERATING SYNTHETIC DATASET**

   As suggested in the Deep Learning Capstone Project, I decided to generate a synthetic dataset by concatenating single digits taken from the MNIST database. The code randomly selects images from the MNIST dataset and concatenates them to create number strings with up to *N* digits. The length of the number string (*N*) can be easily modified on code. For testing purposes, I generated numbers with up to 4 digits.

   To generate different types of images, the script also randomly selects a *shifting factor*, which determines the slope of the number string. The script was designed to generate a homogeneous dataset containing a similar number of images with different string lengths.

Figure 4 shows some of the synthetic images which were generated using the code described above. All the images were saved in grayscale. I subtracted the mean of each image.

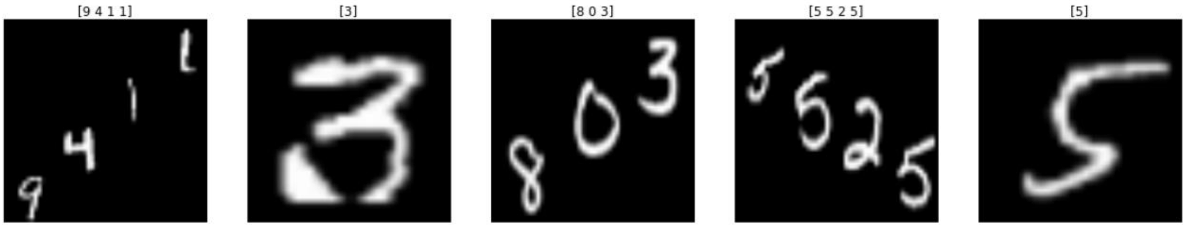

Figure 4 Synthetic images generated by concatenating single digits from MNIST database

2. Preprocessing SVHN dataset (Classification Model)

I performed a very similar preprocessing as de one described by Goodfellow et al. [1]:

- I first found the smallest rectangular bounding box containing all the digits in the respective image. This can be easily done by analyzing the individual bounding boxes provided by the SVHN dataset shown in red to the left of Figure 5.
- I then expanded this bounding box by 30% in both x and y directions, crop the image to that bounding box and resized the crop to 64x64 pixels. One of this expanded bounding boxes is shown in blue to the left of Figure 5.
- After that, I cropped several 54x54 pixel images from random locations within the previous 64x64 image. Figure 5 (right) shows some of the 54x54 crops made over the 64x64 pixels image. Although Goodfellow et al. [1] reported and improvement of only 0.5% using this data augmentation, I believe this to be an important step to achieve good performance with real images.

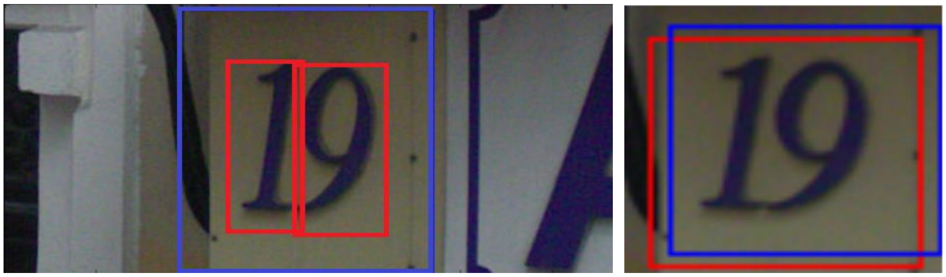

Figure 5 To the left, the individual bounding boxes provided by the SVHN dataset are shown in red together with the expanded bounding box in blue. To the right, the image has been cropped and resized, and 54x54 pixel images are cropped from random locations (shown in red and blue)

All the images are then converted into grayscale to reduce the model complexity. Finally, we subtract the mean of each image and apply random brightness and contrast transformations.

3. Preprocessing SVHN dataset (Regression Model)

I used the SVHN dataset once again to generate training and validation samples for the Regression Model. To guarantee scale and location variance in our training samples, I generated several images by sliding a window through the original images. The stride of the sliding window can be easily modified on code. Figure 6 shows 3 training samples generated from the original image. More than 300000 training images were generated this way.

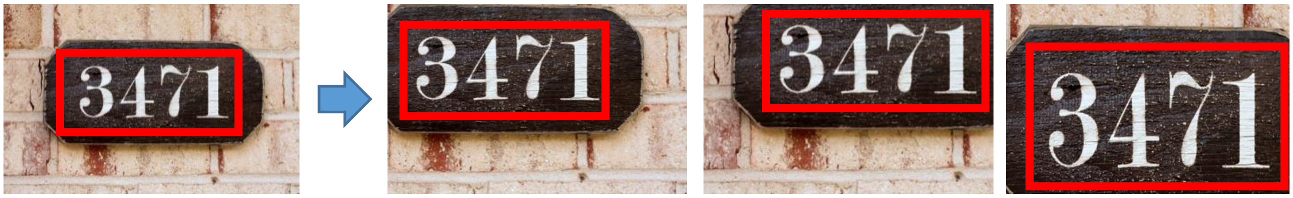

Figure 6 Examples of data augmentation for the Regression Model. Generating several samples by shifting the location of the bounding boxes and changing their scales.

Because our Regression Model also outputs the probability of the image containing a number, we required some "negative" training examples. I employed the CIFAR-10[3] dataset which contains 50000 images of 10 different categories like birds, cats, airplanes and dogs among others. The final dataset was split in training, validation and testing sets.

The Regression Model, in contrast to the Classification Model, employs RGB images and linearly scales them to have zero mean and unit norm (image whitening).

Implementation:

All the code was implemented as Jupyter Notebooks. The code, both for the Regression and the Classification models, was divided into the following steps:

1. Both the validation and test sets are loaded into memory. The preprocessing described before is applied to all samples.

[3] The CIFAR-10 dataset. https://www.cs.toronto.edu/~kriz/cifar.html

2. The training dataset, on the contrary, had to be read dynamically from binary files due to computational constraints. Loading the data this way allowed me to spare a lot of memory during training.
3. Define accuracy function.
4. Define the network architecture.
5. Define loss function and learning method.
6. Train the network logging training and validation loss into Tensorboard.
7. Save the model
8. Plot the performance on some sample images.

Both the Regression and the Classification networks share the same architecture shown in Figure 7 (although the Regression Model employs an additional output). The specific details of these networks will be presented in the following sections.

The loss of the Classification Model is defined as the mean cross entropy error between the outputs and the labels.

The loss of the Regression Model is defined as the mean L2 distance between the bounding box coordinates plus the mean cross entropy error of the confidence score.

Both networks employ an exponentially decaying learning rate. The initial learning rate was tuned for each network individually.

Finally, I employed an Adagrad Optimizer to minimize the loss on both networks.

The training steps can be modified on code and all the parameters are saved periodically during, and at the end of the training. It is possible to run the training code several times to improve the performance of current models.

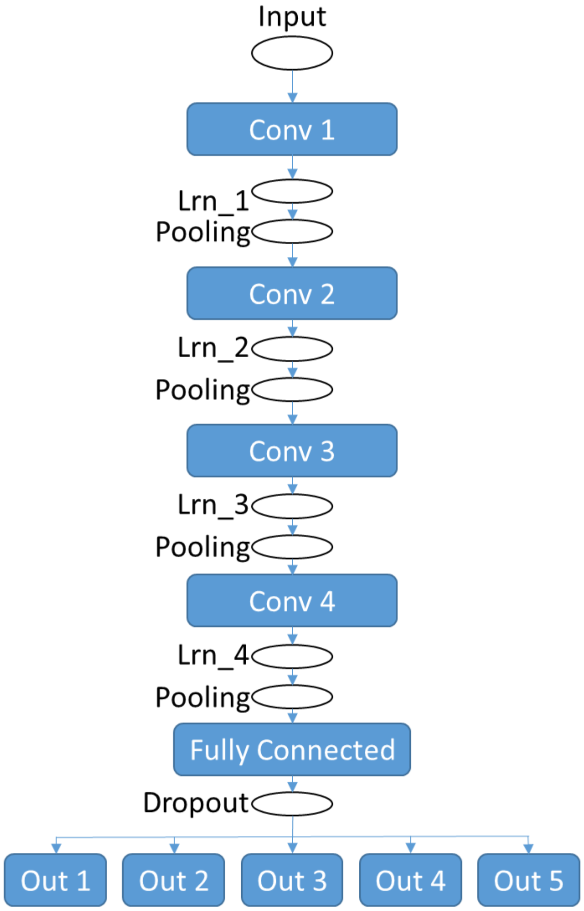

Figure 7 Networks architecture

Once both networks are successfully trained, it is possible to test the system on real webcam images. I have provided a final Jupyter Notebook which uses OpenCV to load camera frames as a video sequence. The original images are resized to 48x64 pixels (I choose these values because my webcam provides 480x640 images and it was therefore possible to avoid distortions) and they are used as input to the Regression Model. If this first network detects the presence of a number within the image, the system crops the image according to the predicted bounding box. This crop is then sent to the second network which converts it to grayscale and which recognize the numbers in the image.

The provided code displays the captured images together with the predicted bounding box (which is scaled to the original image size) and the number which is detected on the top left corner as shown in Figure 8.

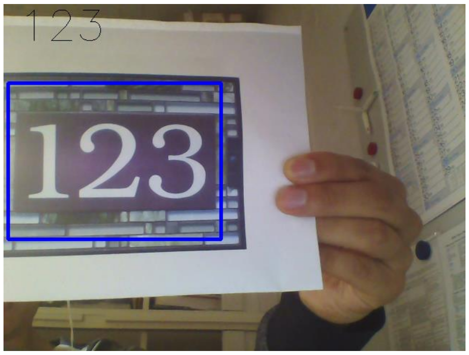

Figure 8 Evaluating the networks performance directly on a webcam frame. The predicted bounding box is shown in blue and the number transcription is drawn at the top left corner.

Refinement:

Initial Configuration
Input (32x32 Grayscale Model)
Conv5-16
Local Response Normalization
Max Pooling
Conv5-32
Local Response Normalization
Max Pooling
Conv5-64
FC-64
Dropout
5 x Soft-Max

I started using the Classification architecture shown to the right. The convolutional layer parameters are denoted as “Conv{receptive field size}-{number of channels}”. The ReLU activation function is not shown for brevity. Max-pooling is performed over a 2×2 pixel window, with stride = 2. The batch size was set to 64 samples.

This initial architecture performs quite well when using the synthetic dataset achieving a validation accuracy of 93.8% after 20000 training steps. The same model, however, performs poorly on the more demanding SVHN dataset, achieving a classification accuracy just above 60%.

In contrast, the model proposed by Goodfellow et al. [1] consisted of eight convolutional layers and significantly more parameters, obtaining a transcription accuracy of 96.03%. Unfortunately, I do not have the computational resources to train such a model.

For this reason, I decided to implement some of the ideas proposed by Simonyan & Zisserman [2] in their VGG architecture. The main idea is that we can replace a large receptive field convolutional layer with a stack of very small convolutional filters. They demonstrated that doing this not only reduces the number of parameters in the model, but also improves the performance by making the decision function more discriminative. As I will show in the following section, I replaced all the 5x5 conv. layers by a stack of two 3x3 layers.

This was, of course, not the only refinement I had to make to improve the performance of the network. The following parameters were also modified in an iterative way trying to optimize the performance on the validation set while avoiding overfitting:

- Number of convolutional layers: I tried models with up to 5 convolutional layers (each one consisting of a stack of 2). Having 5 layers did not improve the performance significantly, but it did the training process much slower. The final model employs 4x2 layers.
- Depth of the convolutional layers and nodes in the fully connected layer: I increased the number of channels on each layer and the size of the FC layer slowly trying to find a good threshold between performance and training time.
- Learning rate: I had some problems when starting the project and it was all due to a large learning rate. The model was overshooting during training and the loss was not decreasing even after training for many steps. It took me some time to find out the reason, but once I did I could find optimal values.
- Dropout: The dropout probability was also tuned after trying a couple of different values.

Results

Model Evaluation and Validation:

As it has been mentioned before, I made use of validation sets to evaluate the performance of both models and to properly configure their architectures. Figures 9 and 10 show a plot of the training and validation losses for both models. The training process was stopped whenever no further improvement was observed in the validation loss/accuracy.

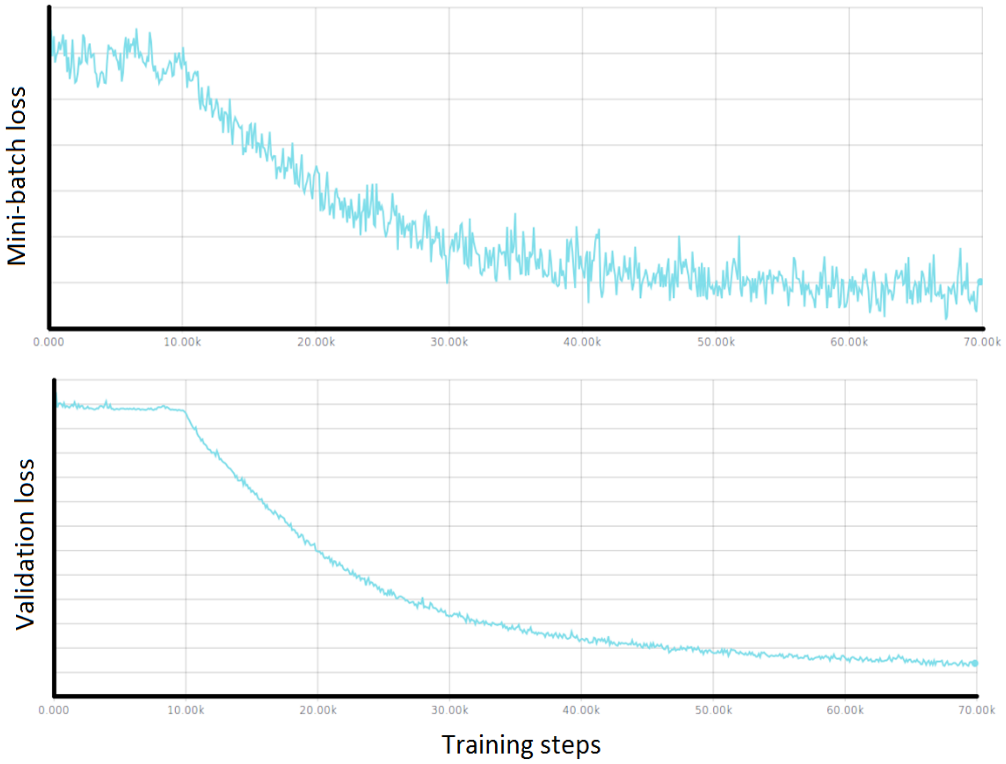

Figure 9 Training and validation loss presented by the Classification model. Training was stopped when no further improvement in the validation set was observed.

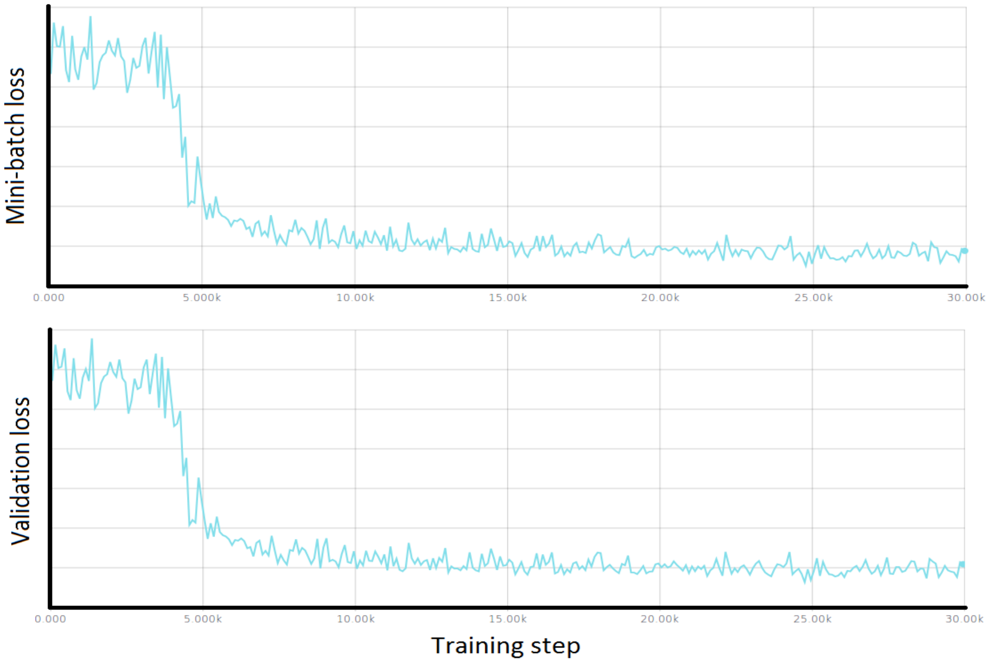

Figure 10 Training and validation loss presented by the Regression model.

The final configurations are presented in the following table, where I use, once again, the notation: “Conv{receptive field size}-{number of channels}”. I trained different models and the ones presented here were the ones which performed the best.

ConvNets Configuration	
Regression Model	***Classification Model***
Input (48x64 RGB Image)	Input (54x54 Grayscale Image)
Conv3-32	
Conv3-32	
Local Response Normalization	
Max Pooling	
Conv3-48	
Conv3-48	
Local Response Normalization	
Max Pooling	
Conv3-64	
Conv3-64	
Local Response Normalization	
Max Pooling	
Conv3-80	Conv3-96
Conv3-80	Conv3-96
FC-2048	FC-3136
Dropout	
[Soft-Max] + [4 x ReLU]	5 x Soft-Max

All the convolutional layers have a stride of one, so that neither of them decreases the resolution of the image. The ReLU activation functions are not shown for brevity. Additionally:

- Max-pooling is performed over a 2×2 pixel window, with stride = 2.
- The batch size was set to 64 samples.
- The dropout probability was set to 0.9375 during training.
- The initial learning rates α_0 were set to 0.002 and 0.015 for the Classification and Regression models respectively, with a decaying factor of 0.95.

The final performance of both models is presented in the next table. It can be seen that the Classification network achieves a transcription accuracy of almost 90% on the testing dataset. Similarly, the Regression network reaches a detection accuracy of about 84% while achieving an average Dice's coefficient of 0.62. I noticed, however, that the SVHN testing samples are considerably more complex when compared to the training ones. For this reason, I also evaluated the network on a separate testing set which was generated using some samples taken from the SVHN training folder; images which of course were removed from training datasets. The Regression model reaches a Dice's coefficient of 0.81 on this additional testing set.

Evaluation Dataset	Classification model	Regression model	
	Transcription accuracy (%)	*Detection accuracy (%)*	*Dice's coefficient*
Training	90	100	0.87
Validation	89.8	85.5	0.62 (*0.82)
Testing	89.2	83.9	0.62 (*0.81)

Justification:

As presented in Figure 11, the robustness of the proposed approach was evaluated using real webcam images. The system localizes and classifies the numbers which are captured by the camera and it is very robust to changes in orientation and scale. Besides, the system is capable of running online, thus making transcriptions directly from video sequences.

The system, however, is far from perfect. There are some cases where the networks fail either to localize the numbers or to correctly classify them. Some of this examples are shown in Figure 12. I noticed that shaking the objects I was holding during the video recordings had a strong and negative effect on the performance. Holding the numbers steadily and making slow movements resulted in a much better localization/classification accuracy.

It is important to note that, although I have employed a relatively simple network configuration (in comparison to state of the art architectures), the performance of the system is quite decent. There are, of course, many ways to further improve the models as it will be discussed in the Improvement section.

Conclusion

Free-Form Visualization:

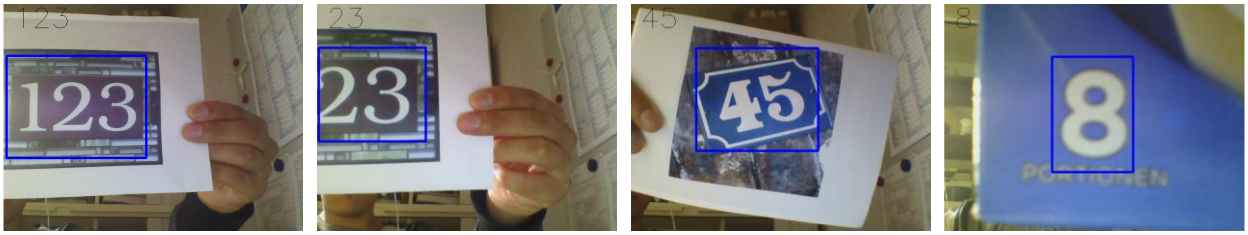

Figure 11 Examples of successfully classified images. The Regression models correctly identifies the location of the numbers and the Classification model is then capable of recognizing the numbers inside the bounding box.

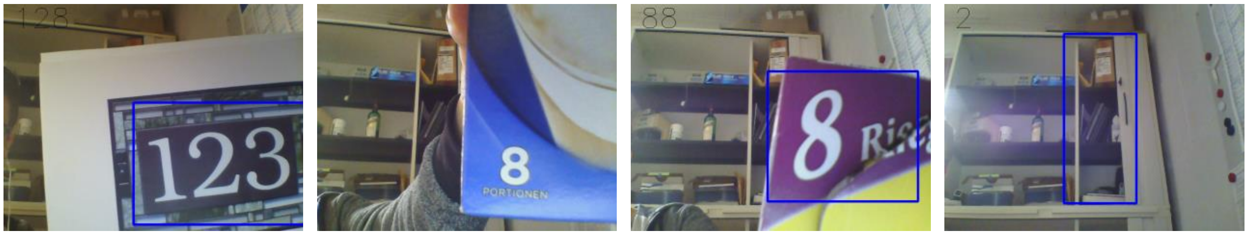

Figure 12 Some examples where the networks fail either to localize or classify the numbers. In the first and third examples (starting from the left) the network correctly localizes the numbers, but it does not classify all the digits correctly. On the second example, the network cannot detect the number and in the last image the network incorrectly identifies a portion of the background as a number.

Reflection:

I have presented a 2-stage approach for detecting, localizing and classifying numbers from image inputs. I use a first Regression network to detect and localize the bounding box surrounding the numbers within the image, and a second Classification network to recognize which digits can be observed inside this bounding box. I have provided some algorithms to preprocess publicly available image datasets like MNIST and SVHN, and I have made use of data augmentation techniques to generate the additional training samples required by the proposed models.

The main reason why I choose to work on the Deep Learning Capstone Project was because of my desire to acquire practical experience implementing deep models and to get the opportunity to use and master libraries like TensorFlow. Because of this, I am very satisfied with the things I have learned while developing this project and very excited to start new and more challenging projects.

Improvement:

It is clear that the proposed solution could be significantly improved by increasing the complexity of the network architectures (make the networks deeper!). This will, however, require many more training samples (to avoid overfitting), computational power and training time.

Nevertheless, there are some things which could be done without drastically changing the current system. I recently read that the pooling layers may decrease the performance of our Regression model, given that most of the spatial information is thrown away. We could redesign the Regression architecture by removing the pooling layers and properly adjusting the convolutional ones.

I also believe that some of the false positives delivered by the current systems are caused by the lack of sufficient “negative” samples during training. Once again, increasing our training dataset could be beneficial.

References

[1] Ian Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, Vinay Shet. "Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks". ICLR, 2014.

[2] Karen Simonyan, Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. ArXiv technical report, 2014.