Udacity Machine Learning Nanodegree
CAPSTONE: Photo OCR Prototype

Wolfgang Steiner
wolfgang.steiner@gmail.com

January 2017

1 Introduction

In this capstone project I present the prototype of a photo OCR (optical character recognition) pipeline based on a sliding window algorithm that is able to automatically detect and parse text in images. Systems such as this are used in a wide variety of applications such as the automatic parsing of house numbers from street view photos [1], automatic translation of text [] or scanning of documents using a mobile device [].

The task of photo OCR can be divided into three distinct stages which are shown in Fig. 1. The first stage of this pipeline is text *detection* which aims to determine bounding boxes for each distinct set of characters in the image. The second stage is character *segmentation*, in which each bounding box is scanned for character gaps with the aim of finding distinct characters that can finally be classified in the third stage by the character *classification*.

Each stage of the pipeline consists of a convolutional neural network

In order to reduce the complexity and training time of the project, the pipeline is confined to only process and parse digits but every stage could be extended and retrained to also handle letters and other characters.

Similar to the work in [2] synthetic images have been generated to train the classifiers.

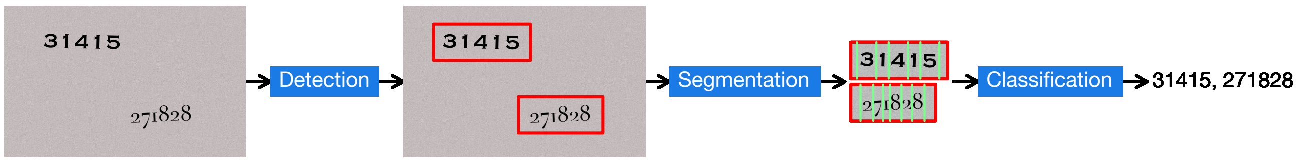

Figure 1: Photo OCR pipeline.

2 Metrics and Benchmark

In order to assess the performance of the complete OCR pipeline, a set of test images x_i containing a randomly generated string of digits of varying length is processed and the extracted string $a(x_i)$

is compared to the real label $y(x_i)$ using the following metric:

$$\text{accuracy}(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \begin{bmatrix} y(x_1) \\ y(x_2) \\ \vdots \\ y(x_n) \end{bmatrix}) = 1 - \frac{\sum_{i=0}^{n} \min\left(\text{lev}(y(x_i), a(x_i)), |y(x_i)|\right)}{\sum_{i=0}^{n} |y(x_i)|} \tag{1}$$

Here, $a(x_i)$ is the label (string of digits) predicted by the system and $|y(x_i)|$ is the length of the i-th true label. $\text{lev}(y(x_i), a(x_i))$ denotes the Levenshtein distance [3] between the true label and the predicted label. The Levenshtein distance counts the number of edit operations that are required to make two strings exactly equal by either deleting, inserting or changing single characters. When calculating the accuracy, the Levenshtein distance is limited to the length of the true label (by use of the min function). This prevents the accuracy of a single example to become negative.

Human level performance on the task of text transcription is often estimated to be greater than 98% [1]. So an obvious benchmark for a OCR system would be to attain comparable or even better performance than a human is capable of. When extracting text the performance of the system can be augmented by use of a dictionary to correct badly classified characters. When dealing with number strings only, this augmentation is not possible and the accuracy of the OCR system

A second performance benchmark is speed. One application for an OCR system would be text extraction of documents scanned by use of the camera of a smartphone. The images would then be send to a server for processing or even better would be precessed directly on the mobile device. In both cases scanning a photo for text should not take more than a few second. For use in an augmented reality app, the speed needs to be even faster in order to allow for an acceptable frame rate. In this case, processing on the mobile device should be faster than about 250ms.

3 Training Data

In this project I predominantly used synthetic images. In order to have a large variety of fonts, I used the google fonts repository as a source [4]. After excluding some symbol and non-latin fonts, a collection of 1525 fonts was available for synthesizing images. The procedure for synthesizing training images is as follows:

- A text and a background color is chosen randomly.
- A background image is synthesized by upscaling noise by a random factor and applying gaussian blurring.
- One of the 1525 fonts is chosen randomly.
- One of the characters is chosen randomly.
- Adjacent characters are added in some images to the left or right.
- An outline is randomly added to the characters.
- A shadow of random width and direction is added to the characters.
- The image is rotated by a random amount.
- Gaussian blur is applied with random radius.

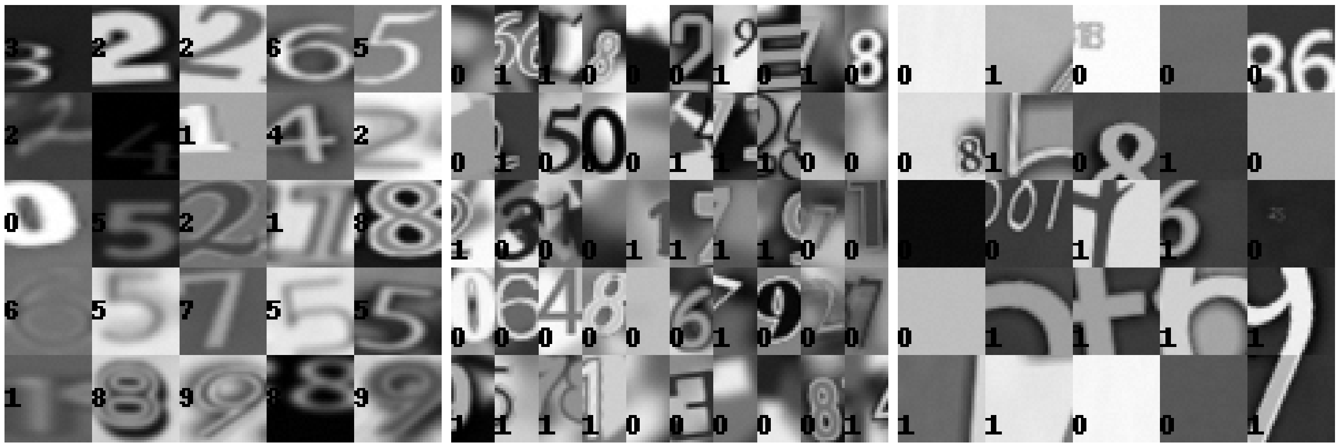

Figure 2: Examples of synthesized images used to train the character classifier (left), character segmentation classifier (center) and character detection classifier (right).

- White noise of random amount is added to the image.

Some examples of images used to train the three required CNN classifiers are shown in Fig. 2. In all cases, training and prediction is done with grayscale images with a fixed height of 32 px. Training images are not pre-generated before training but are generated online during training. For this I implemented python generators for the three types of training images that synthesize batches of images in separate worker threads.

Training images for the character classifier (Fig. 2, left) have characters centered in the images (32x32 px) as will be encountered by the classifier after segmentation. Characters are also resized to the width of the image as the windows between character boundaries are rescaled to the input width of 32px. The labels are simply the corresponding character and are one-hot encoded for training and prediction.

Training images for the segmentation classifier (Fig. 2, center) are only 16px wide in order to improve the spacial resolution of the segmentation procedure (see below). Here, label "1" is assigned if a character starts to the right of the center (for the beginning of a character sequence), a character starts to the left of the center (for the end of a character sequence), or if the spacing between characters is in the center of the image (for segmenting between characters of a sequence). Label "0" is assigned, if no character is present in the image or if a character is in the center of the image.

Finally, the training images for the character detection classifier (Fig. 2, right) are designed to find bounding boxes for character sequences that fit as tightly as possible in order to improve the accuracy of character segmentation and classification. Because of this, images that only have small text areas and images without characters are labeled "0". Only images that have a high percentage of their area filled by character geometry are labeled "1".

4 Algorithms and Techniques

4.1 Text Detection

4.1.1 Sliding Window Algorithm

The first step in the OCR pipeline is text detection. Here a two dimensional sliding window algorithm is used to detect characters in the image and to fit bounding boxes around each character sequence encountered. In order to detect text of different sizes, the process of sliding windows is repeated at different scales. The choice of minimum and maximum scale factors are application specific and directly determine the biggest/smallest size of text that can be recognized and processed by the following stages of the pipeline. When scaling the image for one iteration of the sliding window procedure, it is also rescaled so that its width and height are multiples of 32px, which is the input size of the character detection classifier.

The image data of each window is then concatenated into an input tensor for the character detection classifier which has a sigmoid activation function at its output. If the predicted value for a window exceeds a certain threshold (0.95) it is added to a rectangle list that computes the union of overlapping rectangles. At the end of the procedure, each union of positively classified rectangles is likely to be a bounding box containing a character sequence which is then segmented into single characters for classification.

4.1.2 Character Detection Classifier

4.2 Character Segmentation

4.2.1 Sliding Window Algorithm

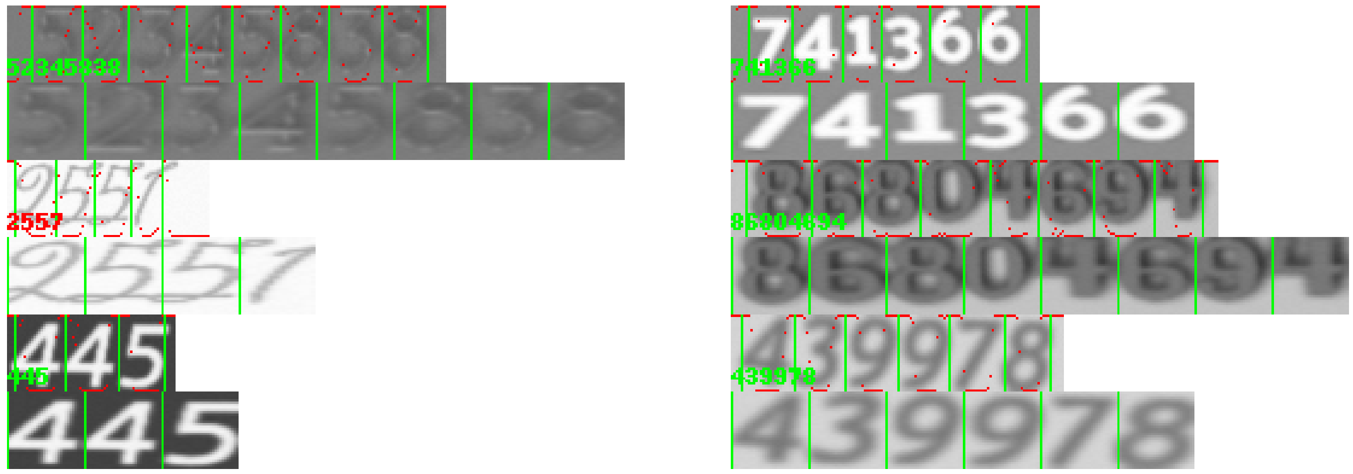

Figure 3: Examples of the character segmentation. The top row of each example shows the input image. The score of the segmentation classifier is visualized in red, with each dot representing the output for one sliding window iteration. The resulting character border is shown in green. The bottom row shows the character images that have been extracted and rescaled based on the segmentation. These images are then fed into the character classifier.

4.2.2 Character Segmentation Classifier

4.3 Character Classification

5 Methodology

6 Results

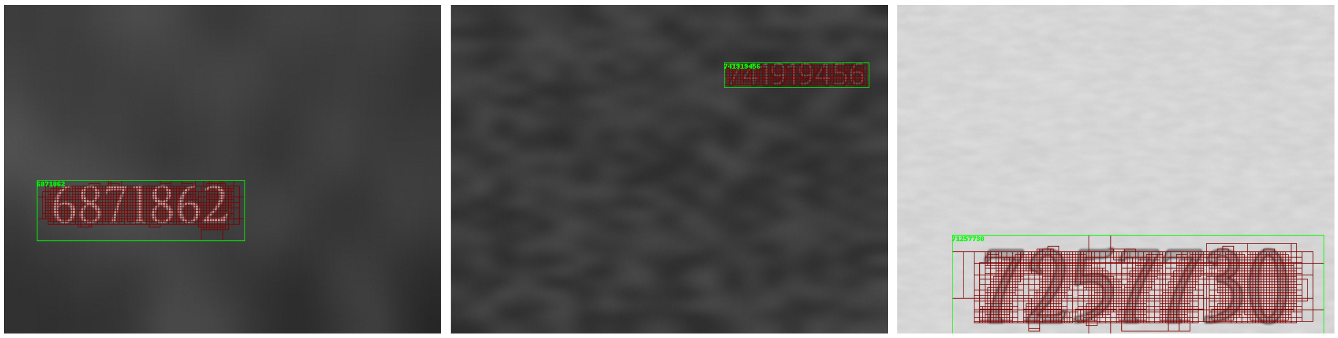

Figure 4: Examples of correctly transcribed digits strings. Red rectangles are positive classifications of the character detection classifier. The green rectangle corresponds to the bounding box calculated as the union of the red rectangles.

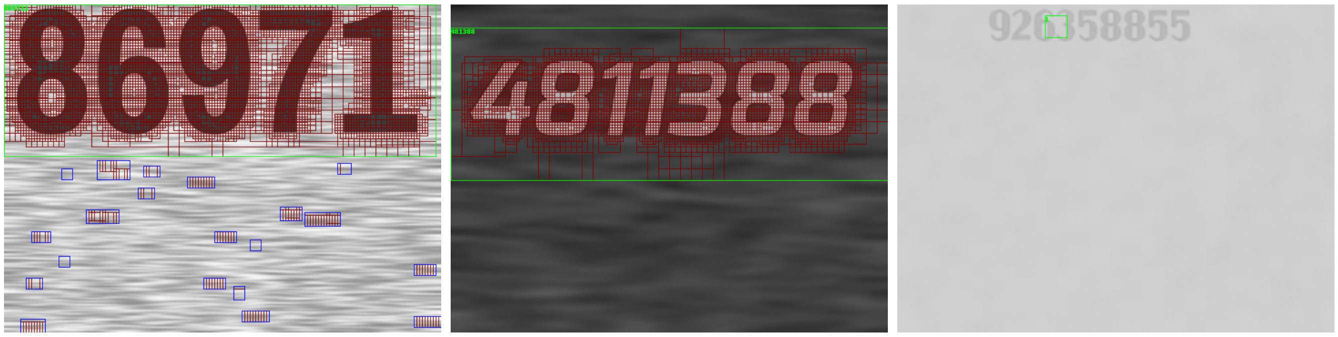

Figure 5: Examples of incorrectly transcribed digits strings. Repetition of digits (left). Missing digits (center). Low contrast images and small test size may result in character failing detection, resulting in a bounding box that is too small for segmentation and classification (right).

As part of this project a proof-of-concept demo was implemented that detects and transcribes strings of digits using a webcam. Example screenshots of this demo in action are shown in Fig. 6. As can be seen, the system works in principle and successfully detects and transcribes digit sequences at different angles of rotation (Fig. 6, (left) and (center)). Regrettably, performance is lacking: even when running the OCR pipeline on a GPU (GTX 1060), the system struggles to deliver an acceptable frame rate. It also struggles when the image is cluttered, as shown in Fig. 6 (right), where an additional edge is introduced. The false positives of the character detector at this edge prevent the system from finding a text bounding box that is tight enough to enable successful transcription of the digit sequence.

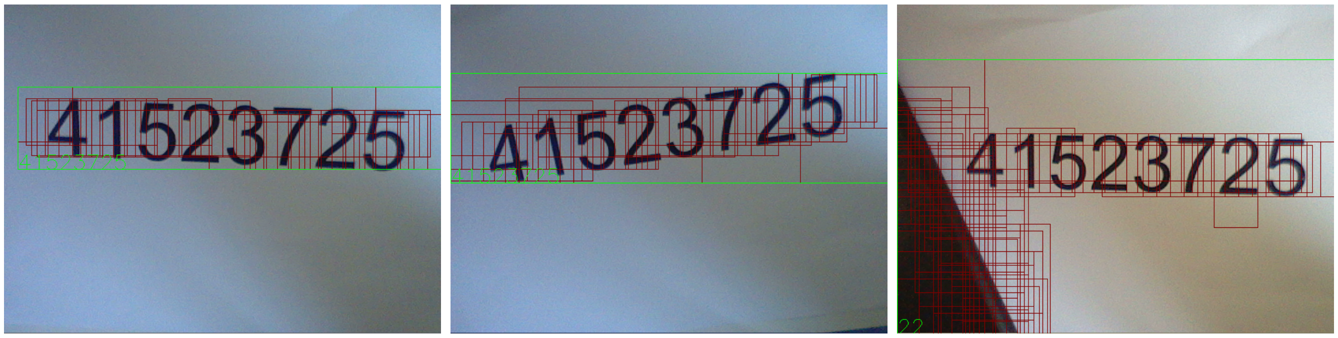

Figure 6: Screenshots of the proof-of-concept implementation of a text detection system using a webcam. Successful text detection (left, center) for different angles of rotation. Failing text detection when an edge of high contrast is visible in the image (right).

7 Conclusion

Implementing a photo OCR pipeline is a challenging problem. Not only do all of the different stages of the pipeline have to achieve a high level of accuracy, their performance also have a direct impact on the later stages. If, for example the text detection stage does not extract a tightly fitting bounding box, the segmentation and classification stages will receive character images that are scaled down vertically, thus reducing their ability to perform at their maximum accuracy.

Accuracy can be improved by increasing the spacial resolution and the number of intermediate scaling steps of the sliding window text detection. Both measures in turn will drastically increases the processing time per image. This problem is further exacerbated if the system should also be able to extract text of small size because in this requires upscaling of the source image prior to text detection.

Apart from the unfavorable accuracy/time trade-off, the sliding window algorithm suffers from an additional drawback: The text detection classifier has to make accurate decisions based only on the contents of a small section of the whole image. All surrounding contextual information is discarded, which makes it very hard to reach a high accuracy. Does an edge detected in a window belong to a character or is it the edge of a piece of paper? Is there an arc of a digit or does the background have some coarse structure to it? These dilemmas either lead to many false positives if the decision threshold is lowered or to incomplete text bounding boxes if the threshold is increased too much which both decrease accuracy in certain applications. Consequently it is very hard to implement a robust system that generalizes well over a broad range of applications and would instead lead to specifically tuned versions for different purposes (e.g. document scanning vs. extracting text from street view images).

Improving the sliding window algorithm could be achieved by a dividing text detection into multiple phases. In [5], candidate bounding boxes are first computed using a simple CNN classifier which are then further filtered on multiple resolutions based on heuristics from the problem domain (car license plate detection). In [2], proposals are filter using a random forrest classifier and then further enhanced using a regression technique.

Overall it seems that the sliding window algorithm, while relatively straightforward to implement (if hard to optimize) and widely used, would better be replaced by a solution that can be trained end-to-end and that takes more image context into account. One technique are R-CNNs

[6, 2], which use a selective search in image space to generate proposal regions that are then fed into a CNN for classification. Another technique is YOLO (You Only Look Once, [7]) that uses a grid of classifiers in order to directly predict the position and size of bounding boxes. Finally, if the (maximum) length of the text is known for the application [1, 5], this will also greatly simplify the detection algorithm as, for example, the aspect ratio of the text bounding boxes can be used to filter out false positives.

References

[1] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet, "Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks," *CoRR*, vol. abs/1312.6, pp. 1–13, 2013.

[2] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Reading Text in the Wild with Convolutional Neural Networks," *International Journal of Computer Vision*, 2016.

[3] "Levenshtein distance - wikipedia." `https://en.wikipedia.org/wiki/Levenshtein_distance`. (Accessed on 01/07/2017).

[4] "google/fonts: Font files available from google fonts." `https://github.com/google/fonts`. (Accessed on 01/08/2017).

[5] H. Li and C. Shen, "Reading Car License Plates Using Deep Convolutional Neural Networks and LSTMs," *arXiv*, p. 17, 2016.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," nov 2013.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CVPR*, 2016.