HAW HAMBURG

Hauptprojekt

Johannes Berger

Building a Prototype for Real-Time License Plate Extraction using Deep Learning

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Johannes Berger

Thema der Arbeit

Prototyp zur Kennzeichenerkennung in Echtzeit mit Hilfe von Deep Learning

Stichworte

Convolutional neural network, YOLO, Automatische Kennzeichenerkennung

Kurzzusammenfassung

Diese Arbeit dokumentiert den Bau eines Prototypen, der es ermöglicht Autokennzeichen in einem Video zu erkennen. Dabei wird ein Faltungsnetzwerk namens YOLO benutzt um die Fahrzeuge zu finden und heuristische Methoden mittels OpenCV, um die Kennzeichen zu extrahieren.

Johannes Berger

Title of Thesis

Building a Prototype for Real-Time License Plate Extraction using Deep Learning

Keywords

Convolutional neural network, YOLO, ALPR

Abstract

In this work the problem of building a prototype system to detect license plates in a video feed is tackled. Inspired by the versatility of deep neural networks the system uses a CNN called YOLO to detect vehicles and heuristic methods in OpenCV to extract the license plates.

Contents

List of Figures

1 Introduction

Measuring the speed of a passing vehicle is a common practice in our society to regulate traffic and ensure road safety. It can be achieved in a variety of ways such as with a radar gun, induction loops or cameras fixed to overarching bridges. All those methods require large, expensive technical equipment which is not trivial to come by or to handle. In this paper a computer vision system is proposed which takes arbitrary video files, analyzes them for vehicles and returns a velocity estimation. It does so by measuring the height of the license plates of the passing cars, an object with known dimensions as they are standardized.

This system is a progression of a much simpler system build for the first masters project (*Grundprojekt*). The system back then was capable of locating license plates within a picture. It did so by finding vehicles with the help of a convolutional neural network (CNN) and then applying various filters and operators on the determined region to precisely locate the license plate.

The new system described in this work improves on that in a variety of ways. The CNN used to detect vehicles is more capable, has a higher detection rate and is still faster. This also shows dramatically in the number of frames the system is able to process per second. The license plate localization was completely reworked and is using a two step approach now. It uses Haar classifiers to propose regions where license plates might be located. This is very fast but results in a lot of false positives. That is why in a second step a CNN is used to validate if a region proposal really contains a plate. This method results in a much higher hit rate compared to the first version but is a the same time a lot more efficient than a sliding window approach. Furthermore the exact dimensions of the detected license plate are now calculated using curve fitting to get the most accurate result. To further reduce the margin of error camera calibration was introduced to correct each video frame for lens distortion.

2 Enhancements

The system has been enhanced in many ways, from a clearer code structure to newer, more powerful technologies. This simplifies further development and improves performance in a major way as will be shown in chapter 3.

2.1 Upgraded Object Detection Network

The first version of this project [1] was build on top of the YOLO [4], a single shot object detector that achieved state-of-the-art recognition scores when it was introduced in 2016. Since then, CNNs have improved significantly and more modern and better performing networks have emerged. One of them is YOLOv3 [6] a progression of YOLO and its successors [5] which is roughly 3 times faster than the original one. The general architecture is similar to the first iteration with some important improvements. YOLOv3 now uses batch normalization which helps to regularize the model and removes the need for the dropout layers. It benefits from higher resolution images for training, helping it detect smaller objects and uses a 13×13 instead of the 7×7 grid to create the initial anchor boxes which are now based on the actual training data and not hand picket any more. It also uses several residual blocks [2] within its total of 56 layers to facilitate the training of this much deeper network.

As I run this project on commodity hardware and want to come as close to real-time performance as possible, it uses a variant of YOLOv3 called tiny-yolo which only uses 20 layers with just one residual block. The number of anchor boxes per grid cell is also reduced from 9 to 6. This comes at a slight reduction in detection accuracy but the inference time per frame is greatly reduced. The performance of this model will be further discussed in chapter 3. Another change in regard to the model is that it was switched from a pure TensorFlow implementation to Keras, an abstraction over TensorFlow which makes the code easier to read and more maintainable.

2.2 License Plate Localization

The license plate localization has been reworked to make it faster and more accurate. The first iteration of this project used a relatively simple approach of finding the plate by using several different filters provided by OpenCV, such as edge detection, opening and closing operations. It then used some simple validations to rule out false positives.

This iteration employs a multi-step approach where regions possibly containing license plates are first selected by a Haar classifier, these regions are then validated by a CNN and the resulting plates are measured with significantly higher accuracy.

2.2.1 Haar Classifier

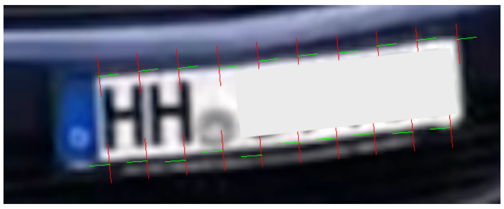
Haar-like features have been around for some time but they are still the tool of choice in specific use cases, mostly because of the ease of computation and the resulting detection speed [7].

To train the classifier I used the existing license plate localization logic with less strict validation rules to search for potential plate regions within frames of various videos. I then labeled those image patches to either contain a license plate or not. With this data set I then trained a 20-stage classifier which is used to propose plate regions.

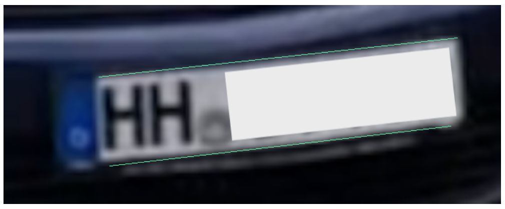
Detailed steps of this process and the accompanying code can be found on GitHub at https://github.com/Xaaris/opencv-haar-classifier-training.

2.2.2 License Plate Validation

The aforementioned classifier is intentionally trained to over-classify regions. That means it produces a lot of false positives but the chance of it missing a plate is relatively low. Thus there is need for a thorough validation step that eliminates all the false positives. As the simple rule based validation from the first iteration of this project was not enough anymore, I instead opted to use a convolutional neural network.



(a) Individual measuring points along the upper and lower edges of the license plate



(b) Final upper and lower lines determined by the algorithm

Figure 2.1: Steps during the height measurement of a license plate (plates masked for privacy reasons)

2.2.3 License Plate Height Measurement

The exact measurement of the license plate height is of utmost importance as it is the reference on which the velocity estimation is based. In the previous section an image patch is obtained which is validated to include a license plate. To measure the height of it the image is first white-balance corrected to get more consistent results. It is then converted to a binary representation based on the intensity of each pixel and undergoes some morphological operations to find a clear outline of the license plate. the upper and lower edges of this outline are then further refined by placing a number of equidistant refinement points on them. For every one of these points the close vicinity in the perpendicular direction to the original line is scanned to determine where exactly the edge of the plate is located with sub-pixel accuracy. A best-fit line is then calculated based on those points. The height of the license plate is then assumed to be the average distance between the upper and the lower refined edges. Figure 2.1 shows two steps from the process where (a) displays the refinement points (10 for each line in this case) and (b) shows the final calculated edges.

2.3 Camera Calibration

Commercially available standard cameras unfortunately come along with one major weakness: significant distortion. Luckily, as the distortions are fixed constants for a given camera model we can easily measure them and correct the resulting picture for them. To be able to do so we first have to measure the camera properties by taking pictures of known patterns, such as a chessboard pattern. By taking multiple pictures from various

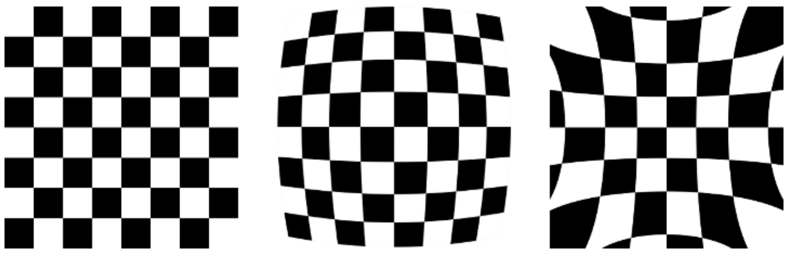


Figure 2.2: No distortion on the left while the middle example shows positive radial distortion and the right example shows negative radial distortion [3]

angles we can then calculate the intrinsic camera matrix with f being the focal length in either direction and c being the principal point, usually at the image center.

$$camera\ matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

This matrix needs to be scaled along the image size in case its dimensions are altered. We can also calculate the rotation distortion

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

as well as the translation distortion in the x and y dimensions.

$$x_{corrected} = x + (2p_1 xy + p_2(r^2 + 2x^2))$$
$$y_{corrected} = y + (p_1(r^2 + 2y^2) + 2p_2 xy)$$

These values are fixed and don't depend on the image scale. With these parameters in place we can correctly translate points from the real world to pixels in a photo. Figure **??** shows how extreme examples could look like. It also demonstrates clearly that we might lose some pixels at the edges of the picture when correcting for the distortion. To not end up with black bars around the edges another matrix is used to crop the image to the region of interest. On an actual example the loss of information in the picture is minimal, so that we do not need to worry that important parts of the image are cropped.

3 Results

FPS doubled Compare tiny yolo to normal yolo

Accuracy: measured 60m with stopwatch at various speeds multiple times (make table with actual and expected speeds) Measured lens factor. focal length not officially documented for video as it crops in from foto

4 Outlook and Conclusion

Bibliography

[1] Berger, Johannes: Grundprojekt: Building a Prototype for Real-Time License Plate Extraction using Deep Learning. (2018)

[2] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Deep residual learning for image recognition. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2016-Decem (2016), S. 770–778. – ISBN 9781467388504

[3] OpenCV: *OpenCV: Camera Calibration and 3D Reconstruction.* 2016. – URL http://docs.opencv.org/master/d9/d0c/group{_}{_}calib3d.html{#}ga549c2075fac14829ff4a58bc931c033d. – Zugriffsdatum: 16.6.2019

[4] Redmon, Joseph ; Divvala, Santosh ; Girshick, Ross ; Farhadi, Ali: You Only Look Once: Unified, Real-Time Object Detection. (2016)

[5] Redmon, Joseph ; Farhadi, Ali: YOLO9000: Better, faster, stronger. In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* 2017-January (2017), S. 6517–6525. ISBN 9781538604571

[6] Redmon, Joseph ; Farhadi, Ali: *YOLOv3: An Incremental Improvement.* 2018. – URL http://arxiv.org/abs/1804.02767

[7] Viola, P. ; Jones, M.: Haar-like. In: *Cvpr* 1 (2001), S. I–511–I–518. – URL http://ieeexplore.ieee.org/document/990517/. – ISBN 0-7695-1272-0