|  |  |
| --- | --- |
| **Capstone Project**  **Machine Learning Engineer Nanodegree** | Andreas Börzel  March 26th, 2020 |

**German License Plate Recognition**

**Definition**

**Project Overview**

Machine learning methods are used today in many different areas. One particularly exciting and important area is machine vision, which is about the ability to understand and interpret image content with a computer. The range of tasks is roughly divided into classification, object detection and localization, segmentation and recognition. Today, machine learning, especially deep learning, makes it possible to detect, identify (identity) and localize objects within an image with the help of a computer, and even to read gestures, feelings or behavioral patterns from images - abilities that until a few years ago were reserved for humans alone.

In this project I created a small Android app that can recognize the license plate of a car quickly and easily with the camera of a smartphone or tablet and translate the license into plain text. The app marks the recognized license plate within the camera image with a bounding box and displays the determined license in plain text as annotation above the bounding box, as outlined in the following demo:



**Problem Statement**

The goal is to create a license number detection and recognition for German car license plates, running on Android devices; the tasks involved are the following:

1. Collect data and create a dataset for license plate detection
2. Train a classifier that can detect and localize a car license plate in an image
3. Create a dataset for license number recognition
4. Train a CRNN Model that can recognize the car license number as plain text
5. Create an Android app that detects license plates with the camera and displays the license number in plain text

The application can be used as a basis for many practical, mobile applications that require the license number of a car as an identification feature.

For simplicity, the project is initially limited to the recognition of German license plates. A later extension to European or even worldwide license plates is possible!

**Metrics**

\*\*\* TODO \*\*\*

In object detection, evaluation is non trivial, because there are two distinct tasks to measure:

* Determining whether an object exists in the image (classification)
* Determining the location of the object (bounding box, a regression task)

The evaluation method used is the “mean average precision” or “mAP score”, this is a commonly accepted evaluation method for object detectors, which has also been used in object detection competitions, such as for the PASCAL VOC, ImageNet, and COCO challenges.

[Here](https://medium.com/@timothycarlen/understanding-the-map-evaluation-metric-for-object-detection-a07fe6962cf3) is a detailed description about the mAP score.

Accuracy is a common metric for binary classifiers; it takes into account both true positives and true

negatives with equal weight.

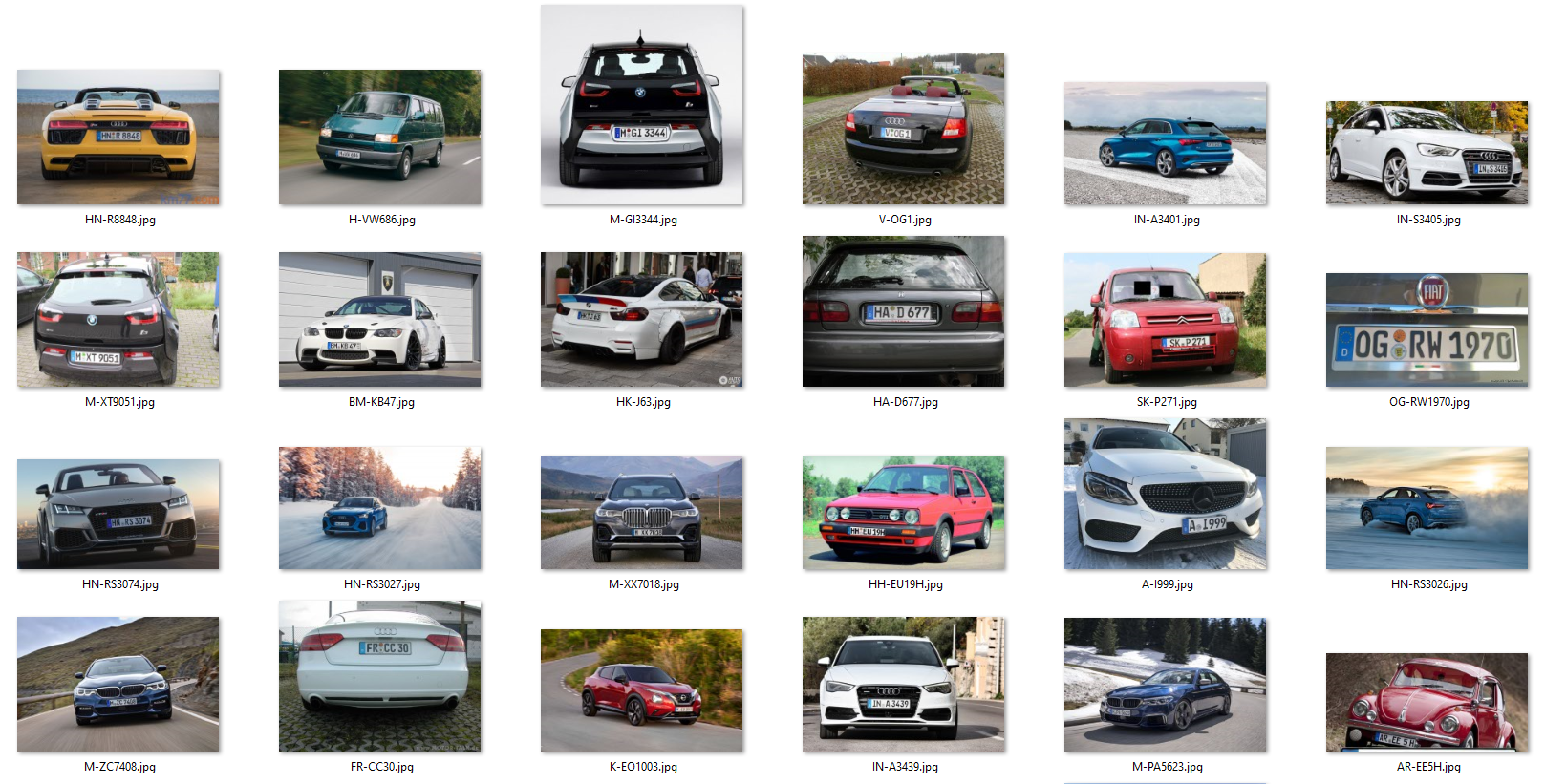
accuracy = dataset size

true positives + true negatives

**Analysis**

**Data Exploration**

**License Plate Detection**

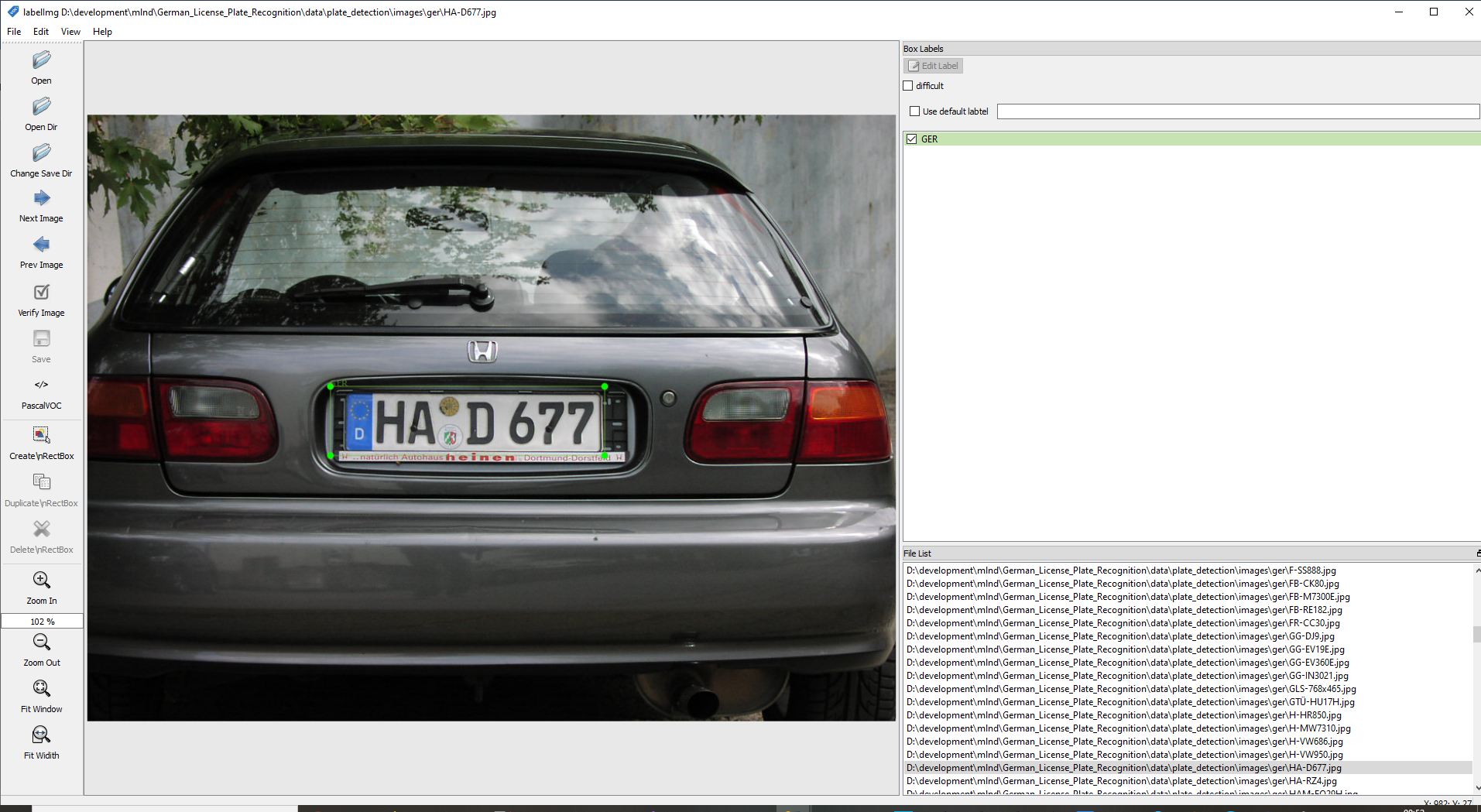
Unfortunately there are no public data collections with pictures of cars with recognizable German license plates. Therefore, a own data collection was created for License Plate Recognition, with pictures collected from the internet, see examples: 

The following criteria were taken into account when selecting the images:

* License number clearly recognisable by human
* Exclusively German number plates
* Sufficient picture quality (minimum resolution)
* Greatest possible diversity (various distances, car brands, colors, environments, weather, license numbers, rotations, …)

Due to legal reasons it is very difficult to get corresponding pictures on the internet, so I could only find 211 pictures which meet the above mentioned criteria.

The captured images were labeled with the tool [LabelImg](https://github.com/tzutalin/labelImg). The car license plates in each image were framed with a bounding box, to which the class “GER” for Germany was assigned. This makes the data collection extendable for license plates of other countries, e.g. IT (Italy), FR (France), ...



LabelImg stores the label information as annotations in the [PASCAL VOC](http://host.robots.ox.ac.uk/pascal/VOC/voc2008/htmldoc/) format. For each image a corresponding XML file is created, which contains the label information like class and bounding box, etc. Here is an example file (HA-D677.xml) for the picture shown above:

<annotation>

<folder>**ger**</folder>

<filename>**HA-D677.jpg**</filename>

<path>**.\data\plate\_detection\images\ger\HA-D677.jpg**</path>

<source>

<database>**Unknown**</database>

</source>

<size>

<width>**1024**</width>

<height>**768**</height>

<depth>**3**</depth>

</size>

<segmented>**0**</segmented>

<object>

<name>**GER**</name>

<pose>**Unspecified**</pose>

<truncated>**0**</truncated>

<difficult>**0**</difficult>

<bndbox>

<xmin>**308**</xmin>

<ymin>**344**</ymin>

<xmax>**655**</xmax>

<ymax>**431**</ymax>

</bndbox>

</object>

</annotation>

From the images and the annotations a data collection in [TFRecord](https://www.tensorflow.org/tutorials/load_data/tfrecord) format was generated, which was used for training the License Plate Detection with the [TensorFlow Object Detection API](https://github.com/tensorflow/models/tree/master/research/object_detection).

The notebook [License Plate Detection Data Exploration And Preparation](https://github.com/aboerzel/German_License_Plate_Recognition/blob/master/1_License_Plate_Detection_Data_Exploration_And_Preparation.ipynb) shows how the data set for License Plate Detection was created.

**License Number Recognition**

To recognize the license plates in plain text, a CRNN model was especially trained, which takes into account the characteristic features of German license plates. Since there were not enough corresponding images available for the training of the model, I decided to create a data collection of generated images and to use data augmentation to create realistic images for the training.

*Structure of German number plates*

Germany is divided into 16 federal states, which in turn are divided into different districts. For each of the districts there are one or more different distinguishing marks consisting of 1 to 3 letters, such as B for Berlin or D for Düsseldorf. There are currently 660 different distinguishing marks:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Autokennzeichen** | **State** | **Stadt/Ort** | **Landkreis/Gemeinde** | **Bundesland** |
| **0** | A | BY | Augsburg | Stadt & Landkreis Augsburg | Bayern |
| **1** | AA | BW | Aalen | Ostalbkreis | Baden-Württemberg |
| **2** | AB | BY | Aschaffenburg | Landkreis & Stadt Aschaffenburg | Bayern |
| **3** | ABG | TH | Altenburg | Landkreis Altenburger Land | Thüringen |
| **4** | ABI | ST | Anhalt-Bitterfeld | Landkreis Anhalt-Bitterfeld | Sachsen-Anhalt |
| **5** | AC | NW | Aachen | Städteregion Aachen | Nordrhein-Westfalen |
| **6** | AE | SN | Auerbach im Vogtland | Vogtlandkreis | Sachsen |
| **7** | AH | NW | Ahaus | Kreis Borken | Nordrhein-Westfalen |
| **8** | AIB | BY | Bad Aibling | Landkreise München & Rosenheim | Bayern |
| **9** | AIC | BY | Aichach | Landkreis Aichach-Friedberg | Bayern |
| **10** | AK | RP | Altenkirchen | Landkreis Altenkirchen (Westerwald) | Rheinland-Pfalz |
| **11** | ALF | NI | Alfeld (Leine) | Landkreis Hildesheim | Niedersachsen |
| **12** | ALZ | BY | Alzenau | Landkreis Aschaffenburg | Bayern |
| **13** | AM | BY | Amberg | Stadt Amberg | Bayern |
| **14** | AN | BY | Ansbach | Landkreis & Stadt Ansbach | Bayern |
| **15** | ANA | SN | Annaberg-Buchholz | Erzgebirgskreis | Sachsen |
| **16** | ANK | MV | Anklam | Landkreis Vorpommern-Greifswald ohne Greifswald | Mecklenburg-Vorpommern |
| **17** | AÖ | BY | Altötting | Landkreis Altötting | Bayern |
| **18** | AP | TH | Apolda | Landkreis Weimarer Land | Thüringen |
| **19** | APD | TH | Apolda | Landkreis Weimarer Land | Thüringen |
| **\* \***  **\*** |  |  |  |  |  |
| **660** | ZZ | ST | Zeitz | Burgenlandkreis | Sachsen-Anhalt |

See [deutsche-autokennzeichen.de](https://laenderkennzeichen.net/deutsche-autokennzeichen/) for complete list.

A car license plate consists of the distinctive mark and an identification number of 1 to 2 letters and up to 4 digits such as AB 1234. Between the distinguishing mark and the identification number is the inspection sticker of the last main inspection (HU) as well as the stamp sticker of the licensing authority with its seal and the state. See [Autokennzeichen.de](https://autokennzeichen.de/kennzeichen-aufbau/).

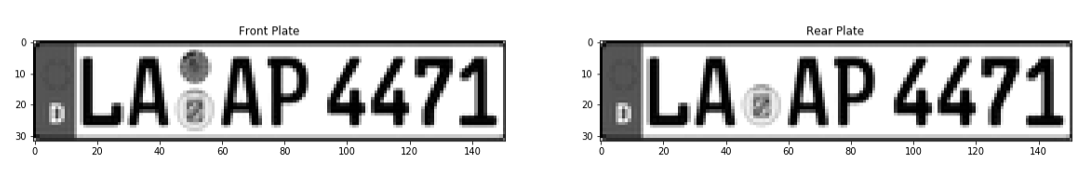
**Example:**



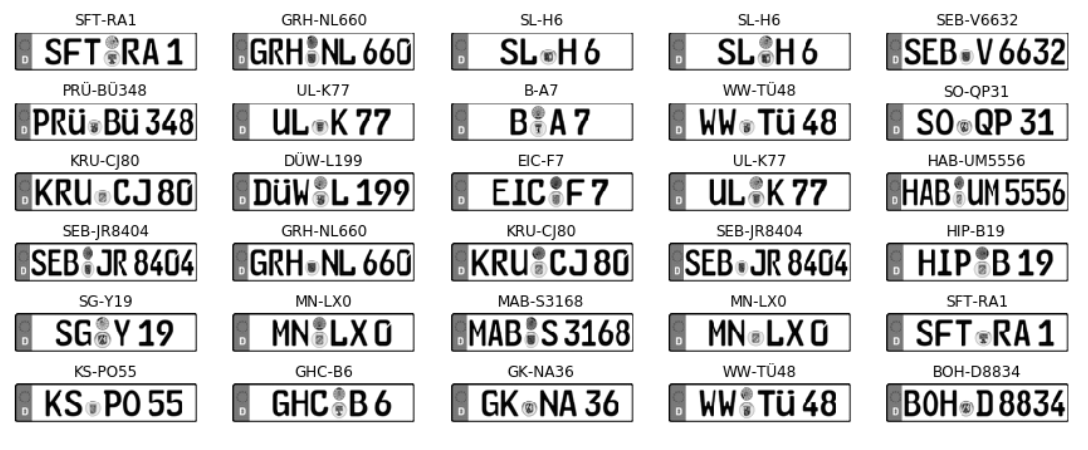
I created a license plate image generator, which generates images of car license plates from randomly generated car licenses, taking into account the regulations for German license plates. The generator uses the [heisnbrg.net](http://nummernschild.heisnbrg.net/fe/) web service to generate an image from a car license number.

The generator observes the following criteria:

* Valid distinguishing marks
* Use only valid characters
* Compliance with format and length restrictions
* Inspection sticker (year/month)
* Stamp sticker of the licensing authority (federal state)
* Differences between the front and the rear license plate

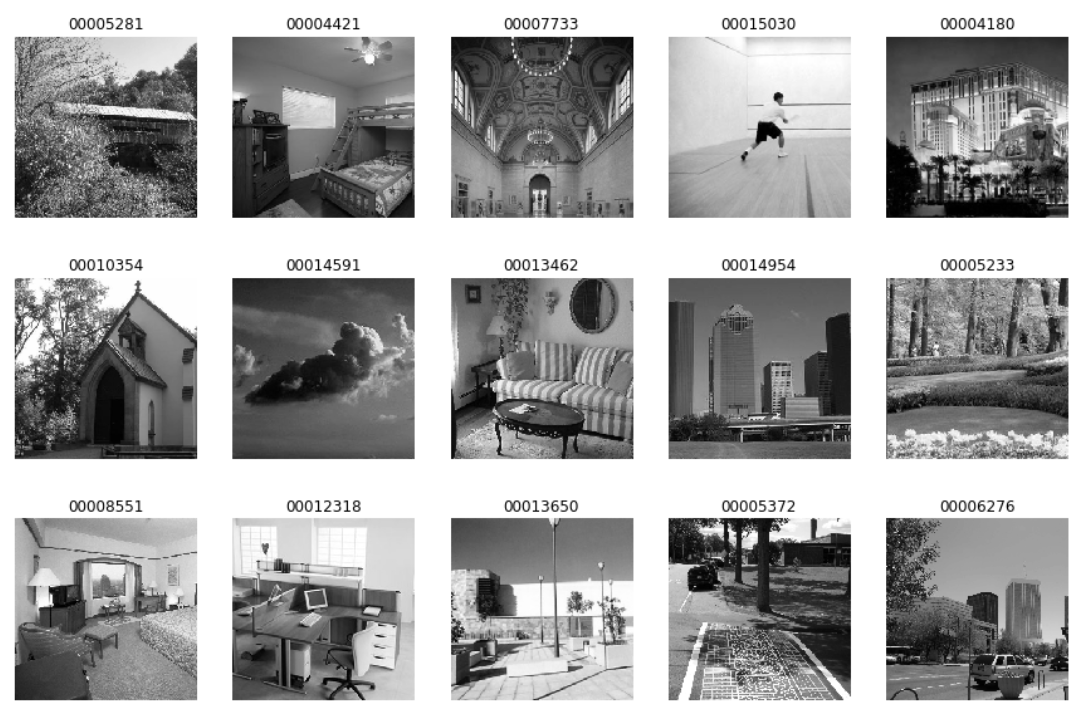
Example of a generated front and rear license plate:

Examples of randomly generated images of car license plates:



For the training of license recognition, 71.533 randomly generated license plates, one image each for the front and rear license plate were generated, i.e. the data collection consists of 143.066 images in total. For easier and faster processing of the images during the training, an [HDF5](https://www.hdfgroup.org/) data set was created from the images together with the labels. This provides easy handling and reduces the I/O load during the training, so that the training process is significantly accelerated.

**Background Images for Data Augmentation**

For the background data collection, 100.000 images were randomly selected from the [SUN2012](https://groups.csail.mit.edu/vision/SUN/) data set and packed into an [HDF5](https://www.hdfgroup.org/) data set. Here are some examples:

The notebook [License Recognition Data Collection And Exploration](https://github.com/aboerzel/German_License_Plate_Recognition/blob/master/3_License_Recognition_Data_Collection_And_Exploration.ipynb) shows how the data set for License Number Recognition was created.

**Algorithms and Techniques**

Number plate detection and number recognition are two different tasks with different requirements. For each of them a model was trained with the help of a method that meets the respective requirements.

The task of License Plate Detection is to recognize German license plates in a camera image (classification problem) and to localize them within the image (regression problem). For this the open source framework [TensorFlow Object Detection API](https://github.com/tensorflow/models/tree/master/research/object_detection) was used, that makes it easy to construct, train and deploy “state of the art” object detection models.

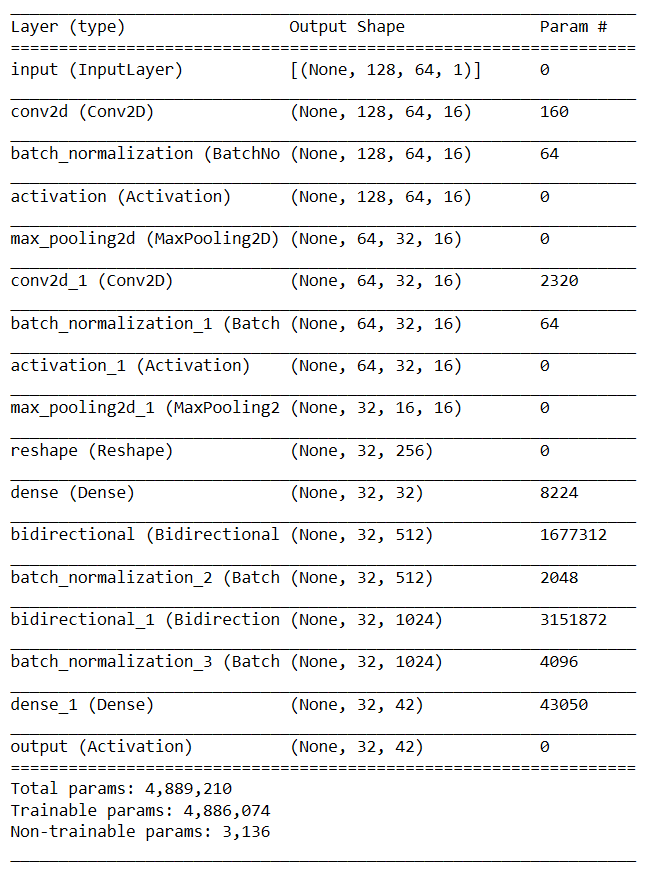
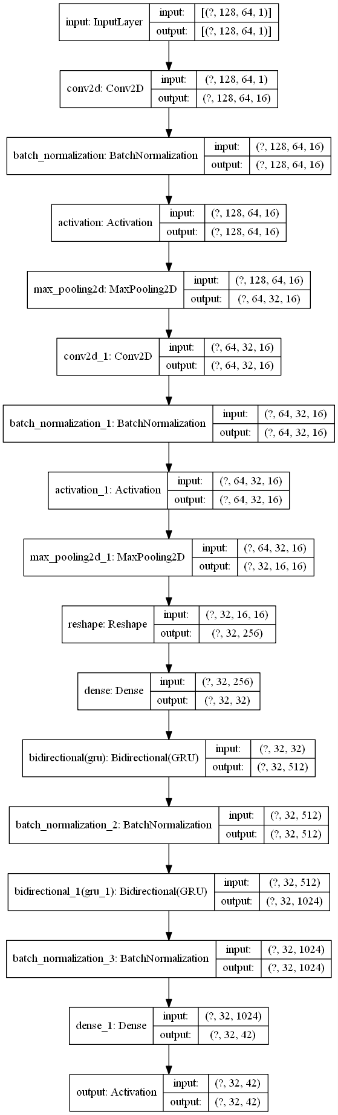
As a starting point I decided to use the [ssdlite\_mobilenet\_v2\_coco](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md) model, an [Single Shot MultiBox Detector](https://arxiv.org/abs/1512.02325) (SSD) which was pre-trained on the [COCO data set](http://cocodataset.org/). This offers a good balance between accuracy and speed, both factors that are important for reliable and user-friendly detection on a mobile device.

Training parameters:

* Training length (number of epochs)
* Batch size (how many images to look at once during a single training step)
* Optimizer (what algorithm to use for learning)
* Learning rate (how fast to learn; this can be dynamic)
* Weight decay (prevents the model being dominated by a few “neurons”)
* Momentum (takes the previous learning step into account when calculating the next one)

See training pipeline configuration [ssdlite\_mobilenet\_v2\_coco.config](https://github.com/aboerzel/German_License_Plate_Recognition/blob/master/config/plate_detection/ssdlite_mobilenet_v2_coco.config).

Task of the License Number Recognition is to recognize the car license number as plain text from the detected license plate image section. I have decided to train a [CRNN](https://arxiv.org/pdf/1911.01577.pdf) (Convolutional Recurrent Neural Network) inspired by the [Keras OCR example](https://keras.io/examples/image_ocr/). The network architecture is as follows:



Training parameters:

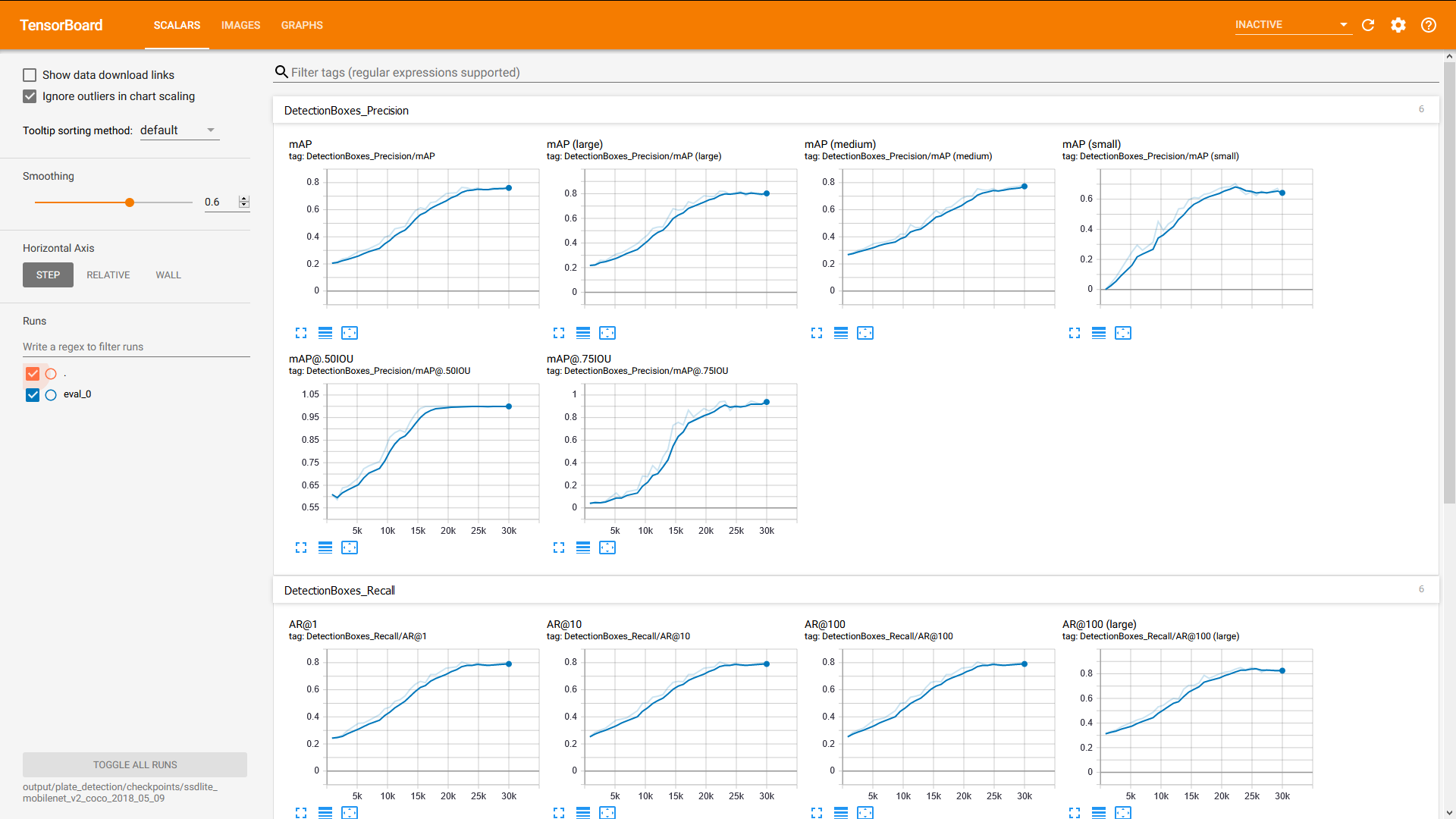
* Training length (number of epochs)
* Batch size (how many images to look at once during a single training step)
* Optimizer (what algorithm to use for learning)
* Learning rate (how fast to learn; this can be dynamic)

[Adagrad](http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf) (Adaptive Gradient Algorithm) is used as [Optimizer](https://int8.io/comparison-of-optimization-techniques-stochastic-gradient-descent-momentum-adagrad-and-adadelta/), this dynamically adapts the learning rate during the training. Furthermore [Early Stopping](https://en.wikipedia.org/wiki/Early_stopping) is used to stop the training if there is no improvement for some time, this reduces the risk of [Overfitting](https://en.wikipedia.org/wiki/Overfitting) and shortens unnecessarily long training time.

Overall it is about [Supervised Learning](https://en.wikipedia.org/wiki/Supervised_learning), i.e. during training the error between the labeled training data (ground truth data) and the model predictions are minimized.

**Benchmark Model**

Since this is a problem of its own and the models are based on data collections that have been created specifically for this purpose, there is unfortunately no possibility of direct comparison with other solutions at first. For evaluation the “mAP score” was considered in the License Plate Detection and the “validation loss” in the License Number Recognition.

Monitoring the mAP score with Tensorboard:

The achieved mAP score for License Plate Detection is 0.76. Considering the few images available for the training and the partly bad image quality this is very good and sufficient for the use case.

The goal was that the Android app should provide the user with a fluent and equally reliable License Number Recognition. For this purpose, the Android App was subjected to intensive practical testing under real conditions.

**Methodology**

**Preprocessing**

The pre-processing steps for License Plate Detection are done by the [TensorFlow Object Detection API](https://github.com/tensorflow/models/tree/master/research/object_detection) out-of-the-box, so there was nothing else to do here.

This is different with License Number Recognition, where realistic images for the training are generated from the generated license number images during the pre-processing step using data augmentation.

Data augmentation steps:

* Varying background (background contains no features)
* Random rotation in X,Y,Z direction (different camera perspectives)
* Varying scale (different camera distances)
* Varying brightness (different times of day and lighting conditions)
* Slight interference effects (Blur and shake of the camera)

Here are some examples of how realistic images are randomly generated from the generated number plate images:



Further pre-processing steps are:

* Image resizing to the network input size
* Normalizing image data between 0.0 and 1.0

**Project Design and Implementation**

The implementation is roughly divided into 3 main tasks:

1. Training of a model for License Plate Detection
2. Training of a model for License Number Recognition
3. Implementing of an Android app that uses the trained models to recognize car license numbers using the camera

**Training of a model for License Plate Detection and Localization in images**

Implementation steps for License Plate Detection:

* Download the pre-trained model from Tensorflow [detection model-zoo](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md)
* Train a finetuned model for License Plate Detection and Localization using the [TensorFlow Object Detection API](https://github.com/TensorFlow/models/tree/master/research/object_detection)
* Export the finetuned model as [TFLite](https://www.tensorflow.org/lite/) model that can be used by the Android app
* Test the trained License Plate Detection model

The implementation of these steps was done in notebook [License Plate Detection Model Training And Evaluation](https://github.com/aboerzel/German_License_Plate_Recognition/blob/master/2_License_Plate_Detection_Model_Training_And_Evaluation.ipynb)

**Training of a model for License Number Recognition from license plate images**

Implementation steps for License Number Recognition:

* Download the License Plate Recognition data set (optional if not already done)
* Load and split the data into a train, validation and test set
* Create a LicensePlateImageAugmentor instance for data augmentation
* Create data generators for the train, validation and test set
* Create the License Number Recognition model to be trained
* Train and evaluate the License Number Recognition model ([Keras](https://keras.io) model)
* Convert the trained Keras model into an [TFLite](https://www.tensorflow.org/lite/) model, which can be used by the Android app
* Test the trained License Number Recognition ([TFLite](https://www.tensorflow.org/lite/)) model

The implementation was done in the notebook [License Recognition Model Training And Evaluation](https://github.com/aboerzel/German_License_Plate_Recognition/blob/master/4_License_Recognition_Model_Training_And_Evaluation.ipynb)

The notebook [License Recognition Workflow](https://github.com/aboerzel/German_License_Plate_Recognition/blob/master/5_License_Recognition_Workflow.ipynb) demonstrates the complete workflow of license plate detection and the subsequent recognition of the license number, as it is done in the Android app.

**Android App Implementation**

The Android app was developed based on the Tensorflow Lite [Object Detection example app](https://github.com/tensorflow/examples/tree/master/lite/examples/object_detection/android). The application was extended from pure object detection to the detection of license plates and the recognition of the license number as plain text and underwent a major refactoring.

The most important classes are:

* [ClassifierActivity.kt](https://github.com/aboerzel/German_License_Plate_Recognition/blob/master/android/app/src/main/java/org/boerzel/glpr/ClassifierActivity.kt) (Calls license plate detection and license number recognition for the current camera image and draws the determined license number onto the image.)
* [PlateDetector.kt](https://github.com/aboerzel/German_License_Plate_Recognition/blob/master/android/app/src/main/java/org/boerzel/glpr/tflite/PlateDetector.kt) (License plate detection and localization)
* [LicenseRecognizer.kt](https://github.com/aboerzel/German_License_Plate_Recognition/blob/master/android/app/src/main/java/org/boerzel/glpr/tflite/LicenseRecognizer.kt) (License number recognition)
* [Detection.kt](https://github.com/aboerzel/German_License_Plate_Recognition/blob/master/android/app/src/main/java/org/boerzel/glpr/tflite/Detection.kt) (Stores score, location and license number of a recognized license plate)

As development environment [Android-Studio](https://developer.android.com/studio) and the programming language [Kotlin](https://kotlinlang.org/) was used.

**Results**

**Model Evaluation and Validation**

During development, a validation set was used to evaluate the model.

**Justification**

Using a Galaxy Note 4 on a 4G network/Wi-Fi, I got the following results:

* The classification delay is about 3 seconds, which is about the same as that of the benchmark
* The processing delay is around 5 seconds, which is better than that of the benchmark
* The per-image classification accuracy7 is higher than 90%8

To understand how successful the final application is, it’s also important to know the overall text

extraction performance, which is illustrated by Figure 7 .

It can be seen that the application is useful for reading labels, but also that it can get confused by hard to read text (such as that on the cardboard box), and it can miss text that is too low-contrast compared to its background.

In summary, the application is useful in a limited domain, but to solve the bigger problem (giving

visually impaired people access to written information), different hardware will have to be used (see the Improvement section)

**Conclusion**

**Free-Form Visualization**

Sdssd

**Reflection**

The process used for this project can be summarized using the following steps:

1. An initial problem and relevant, public datasets were found
2. The data was downloaded and preprocessed (segmented)
3. A benchmark was created for the classifier
4. The classifier was trained using the data (multiple times, until a good set of parameters were found)
5. The TensorFlow Android demo was adapted to run the classifier
6. The application was extended so that it can extract text from images using the Google Cloud Vision API
7. Feeding the extracted text to the TTS system was implemented

I found steps 4 and 5 the most difficult, as I had to familiarize myself with the files of the TensorFlow

Android demo, which uses Bazel and the Android NDK, both of which were technologies that I was

not familiar with before the project.

As for the most interesting aspects of the project, I’m very glad that I found the COCO and

COCO-Text datasets, as I’m sure they’ll be useful for later projects/experiments. I’m also happy about

getting to use TensorFlow, as I believe it will be the deep learning library in the future.

**Improvement**

To achieve the optimal user experience, using more capable hardware9 and moving the text extraction process from the cloud to the device would be essential. This would reduce the processing time and give access to the outputs of all of the modules of the text extraction pipeline, which would, in turn, enable the following features:

* User-guided reading (e.g. read big text first, or read the text the user is pointing at)
* Better support for languages other than English
* Output filtering (e.g. ignore text smaller than some adjustable threshold)
* Passive text detection (auditory cue on text detection, perhaps with additional information encoded in the tone and volume)

The user experience could also be improved significantly by using MXNet, which is a deep learning

library that is better optimized for mobile devices than TensorFlow. The speedup wouldn’t be enough for running text extraction on the device, but it would reduce the classification delay significantly.