Licence Plate Detection

Julien Lamparter 4258560
Muhammed Ömer Akgeyik 6042277
Luca Alfonzo 4176008
Valentin Biermaier 5600687

March 2024

1 Problem Description

This project focuses on developing machine learning models that can recognize and read license plates while also identifying their countries of origin. It involves constructing a comprehensive training dataset for country recognition, training machine learning models to detect license plates and country identifiers, integrating Optical Character Recognition (OCR) for text extraction, and implementing the detection system within a web application.

2 Fundamental Concept

To address this issue, we sought a reliable and efficient solution. Hence, various approaches were considered for accurately predicting license plate details, including the characters and country of origin. One potential strategy involved developing a single model capable of predicting both the license plate and its country, alongside a separate model for extracting the characters. However, this approach posed significant challenges. A primary challenge was the shortage of datasets containing labeled information for both license plate countries and bounding boxes. Constructing such a dataset would require enormous time investment for the model to effectively learn from it.
An alternative method involved using a model to estimate the license plate. Following this prediction, another model would be employed to extract the characters and numbers. Subsequently, a decision tree would be employed to determine the country indicated on the license plate. However, this approach proved less viable due to the necessity of extremely accurate optical character recognition, which could be compromised by poor-quality frames in real-time scenarios. Consequently, the decision tree would frequently generate incorrect country predictions. Even though there are some frames which could predict right, there is the difficulty to obtain the right one. These challenges led to the development of a more effective final approach, suitable even for real-time application with the assistance of a GPU.

The final approach uses a three step strategy, that is shown in the following pipeline:

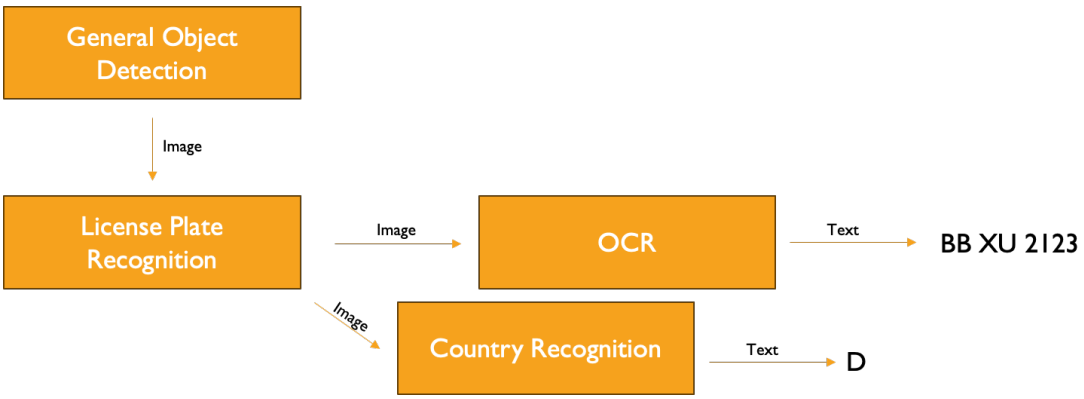

Figure 1: Fundamental pipeline

The process begins with an initial phase of object detection, utilizing the pre-trained YOLOv8s model. Upon the identification of a vehicle within the image, the relevant section is extracted and subsequently processed through the YOLOv8n model, which has been specifically trained for license plate recognition. This model effectively isolates and extracts the license plate from the image. Following this extraction, the license plate image is then passed to two further stages of analysis. Firstly, it undergoes a process of country recognition using Mobilenetv2, self trained model for this purpose. Secondly, the image is analyzed through Optical Character Recognition (OCR) techniques to identify and interpret the alphanumeric characters that are on the license plate. To make it more clear, the following section will provide visual representations of these steps.

Figure 2: Vehicle detection

Figure 3: License Plate detection and OCR

Figure 4: Country detection

We also conducted training on an alternative model for comparison purposes. The second model we employed was SSD Lite with MobileNetV3 Large as its backbone architecture. However, in subsequent analysis, it was found that the YOLO model consistently outperformed the SSD Lite model, a topic that will be discussed in detail later.

Moving forward, the Optical Character Recognition (OCR) process involves taking the larger image containing the identified license plate as input and then attempting to interpret the numbers and letters imprinted on it. To accomplish this task, we used the easyocr library, which offers a robust optical character recognition mechanism.

In the final stage, we employed an image recognition model specifically trained to identify the European country represented on the license plate. To streamline the process for the model, it focuses only on the blue strip of the license plate, where the country code is typically displayed.

Detailed explanations of the steps and models will be provided in the subsequent sections for clarity and precision.

Note: In the later stages of the project (after the final presentation), we decided to shift from using OCR for character recognition to a trained machine learning model for achieving a better performance recognizing characters in license plates. All steps of the pipeline remain the same, except for the substitution of OCR with a trained machine learning model.

3 Dataset and Analysis

3.1 Licence Plate Detection

One of the most important part of constructing a model is the dataset which will be trained on it. We set some goals which our model should predict on most of the cases with a high confidence. One of those goals is that in normal weather terms and a good quality video or image the model should definitely find the bounding box of the licence plate.

To train a model that is able detect license plates in images in realtime, we choose to train on a dataset that contains 15639 train-, 1210 valid- and 624 test-images (1).

Possible train and valid image sets look like the following:

Figure 5: Training set

Figure 6: Validation set

3.2 Country Recognition

Due to the lack of datasets containing only the blue strip of the licence plate, there was a need to create one. For that we used some of the images from the already containing dataset and some images picked from a website containing images with licence plates from different countries from this website: https://platesmania.com/de/. We took specifically different images with different angles and brightness. To cut every image with a tool would consume enormous time. So we created a pipeline in which the blue strip of the licence plate were cutted and saved automatically.

The pipeline was:

1. load the image in the YOLO model and the YOLO model returns a bounding box for the licence plate

Figure 7: Image in dataset

2. Then with the coordinates of the bounding box and 25 pixels wider and higher on addition to that generate an enlarged image of the licence plate. The consideration of 25 pixels is important because sometimes the model does not include the whole licence plate. In the picture 8 you can see in blue the bounding box of the YOLO model and in red the additional 25 pixels.

Figure 8: Image with bounding box

Figure 9: Enlarged image of the bounding box

3. Cut 1/4 of the image to get the blue strip:

Figure 10: cutted enlarged image

After this we used Roboflow to make the annotations as well as the split to train and validation. The final dataset contains 233 train and 62 validation images with 11 classes (countries). The supported countries are:
Belgium, Bulgaria, Germany, Spain, France, Greece, Croatia, Italy, Netherlands, Portugal, Poland.

The amount of countries supported will grow in time.
So there is approximately 21 images in every class. To get more training data and achieve regularization we used the following augmentations while training:

1. Random Rotation between -25 and 25 degrees
2. Brightness saturation and hue transformation
3. Blurring every second image
4. Normalization

With these augmentations the dataset was good enough to train a model only to specify which country on the blue strip is.

3.3 Character Recognition

For character recognition we used a dataset (2) containing 1617 train, 144 validation and 36 test images. In this dataset there are some augmentations already done. The augmentations are a shear of ±15° horizontal and ±15° vertical as well a noise up to 4% of pixels. The images in the data are near to the licence plate and have the following classes: -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, EUR, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z. The '-' do not stand for the character line in the licence plate, it rather stands for not recognizable. So we have an overall of 38 classes and one background class making a total of 39 classes. The classes are not balanced:

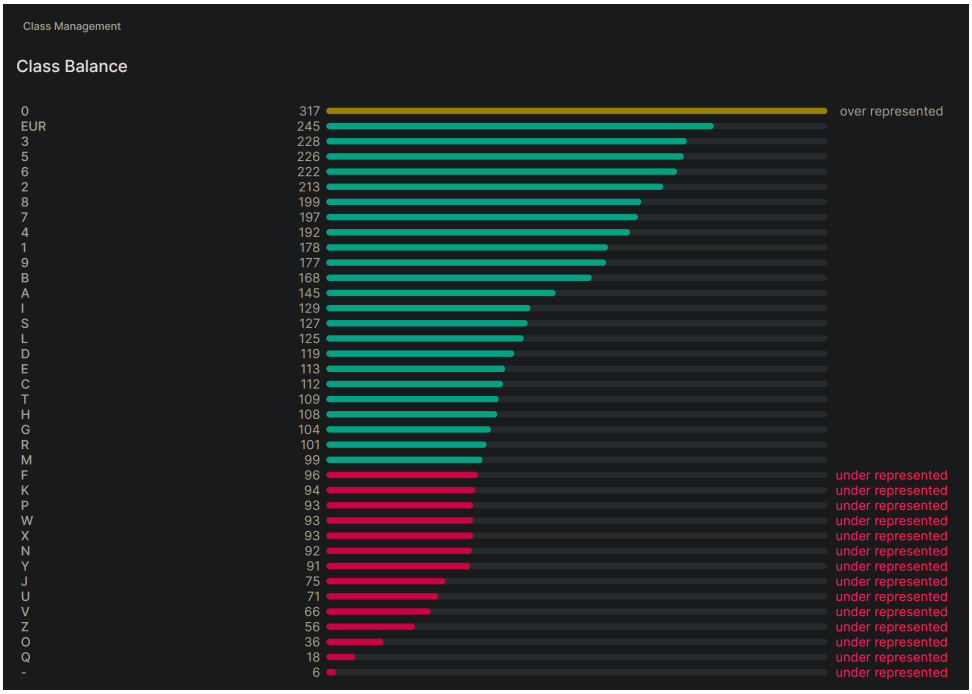

Figure 11: Class balance Character Recognition (2)

4 Architecture and Metrics for Licence Plate Detection

4.1 YOLO

The YOLO (You Only Look Once) object detection algorithm is a great method in real-time image pro-

cessing, designed for speed and efficiency. Initially, YOLO takes an image as input and divides it into a grid of cells, with each cell responsible for detecting objects within its boundaries. The grid's size is determined by the size of the image and the network's last convolutional feature map. A key step involves feature extraction, where each cell is processed through a convolutional neural network (CNN) that has been pretrained on a vast dataset of images to learn features critical for object detection.
Each cell is assigned an objectness score using logistic regression to predict the probability of an object's presence. For cells predicted to contain an object, YOLO determines the class of the object and calculates the probability of the object belonging to each class using a softmax function. In addition, YOLO predicts a bounding box for each detected object, represented by its center coordinates, width, and height, in relation to the cell size.
To enhance the accuracy and remove redundant bounding boxes, the algorithm applies non-maximum suppression, which discards overlapping boxes with lower confidence scores. The final output of YOLO is a list of bounding boxes, each with an associated class and confidence score, indicating the detected objects in the input image.
What sets YOLO apart from other object recognition algorithms is its single-pass approach and grid division method, making it significantly faster. Unlike algorithms that utilize a sliding window approach and are computationally expensive, YOLO processes the entire image in one go. The latest version, YOLOv8, is known for its lightweight architecture that focuses on speed and efficiency, and it achieves state-of-the-art results on various benchmarks. YOLOv8 introduces features like custom anchor boxes and transfer learning, facilitating easier training and customization for specific tasks (3).

In conclusion, YOLOv8 is perfect for real-time object detection. Due to this fact, we choose to use YOLOv8n for the license plate detection and trained it to detect license plates in images. The model size is sufficient as shown in 4.1.1 and also the inference speed of approximately $80ms$ that YOLOv8n provides, is optimal for the use case of detecting license plates, as it aligns with the criteria for real-time processing.

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Figure 12: YOLOv8 model comparison (3)

4.1.1 Training and Evaluation

The training of the YOLOv8n model was conducted over a span of 100 epochs using the dataset specified in section 3. The outcomes of the training process have proven to be very good. The model demonstrates a notable proficiency in accurately detecting license plates in different images. This efficiency can be attributed to the good performance metrics observed, including low precision loss and a high mean average precision (mAP). These metrics are indicative of the model's robustness and reliability in executing the task of license plate detection as shown in figure 13.

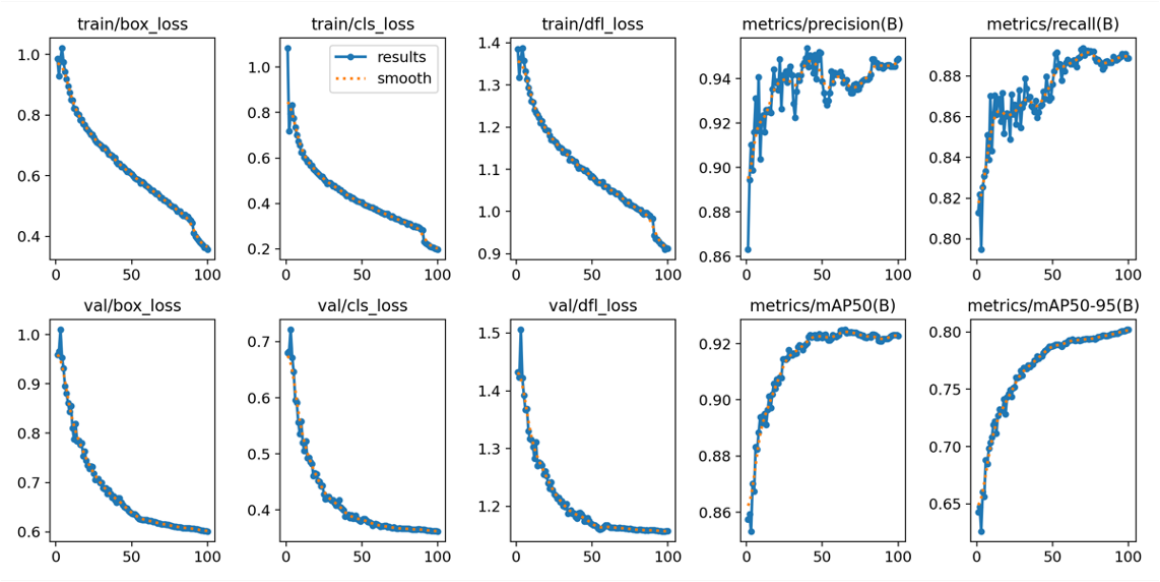

Figure 13: YOLOv8n training results

Taking a closer look at the resulting normalized confusion matrix, we can see that the model is reliably able to detect license plates in provided images.

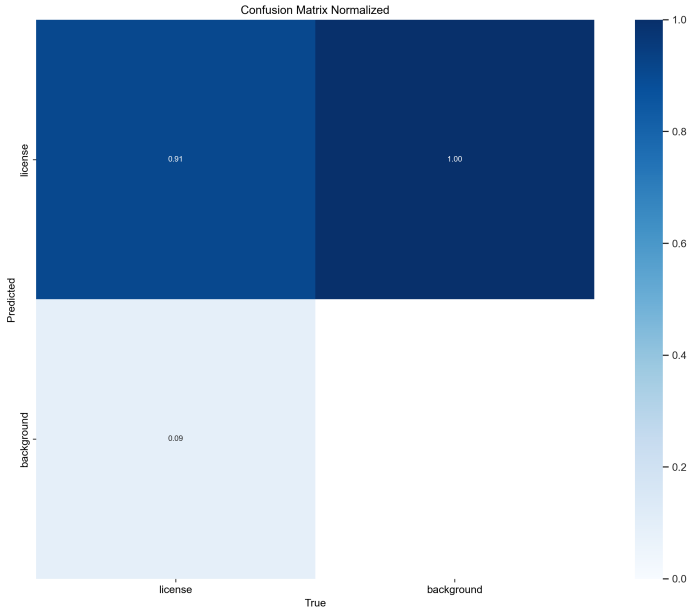

Figure 14: Normalized confusion matrix

4.2 SSD Lite with MobileNetV3 Large

SSD has two main parts, one of them is the backbone while the other one is the head. The backbone serves as a feature extractor functioning as a pre-trained image classification network. In our model the backbone is the MobileNetV3 Large. So the backbone is capable of extracting semantic information from the input image while maintaining the spatial structure at a reduced resolution.
The SSD head includes additional convolutional layers integrated with the backbone. These layers generate outputs used to identify the spatial positions of bounding boxes and object categories.
There are some important concepts introduced in SSD that set it apart from other architectures.
One concept is the grid cell. By dividing each image into a grid, SSD assigns each grid cell the responsibility of detecting objects within its specified region. If an object is present within a grid cell, SSD outputs predictions regarding the object's position, shape, and class. Conversely, if no object is detected, the grid cell specifies the region as background.
The second important concept is the concept of anchor boxes. If we consider having multiple objects in one grid the concept of grid cell in not enough to find more than one object. Anchor boxes solves this with having multiple pre-defined anchor boxes assigned to every grid cell. Those anchor boxes vary in size and aspect ratio, enabling to adjust variations in object appearance. While Training, each anchor box is assigned with positive or negative depending on the IoU threshold compared to the ground truth box. An anchor box with an IoU > 0.5 gets assigned positive. With that there are many anchor boxes being negative.
The third important concept is hard negative mining. After assigning positive or negative to the boxes or in other words after matching there are many negative assignments leading to poor training performance. Therefore this concept involves prioritizing negative examples with the highest confidence loss so that there is a ratio of three negative boxes and one positive box. To conclude SSD's innovative approach to object detection, incorporating grid cells, anchor boxes, and hard negative mining, underscores its effectiveness in accurately identifying objects within images.(4)
The difference between SSDLite and SSD is that SSDLite's architecture complexity is a bit simpler. SSD Lite simplifies SSD's original model heads by employing separable convolutions instead of regular ones. It replaces the heads with new ones that apply 3x3 depthwise convolutions and 1x1 projections (5).

4.2.1 Training and Evaluation

So we see YOLO has a solid mAP and Average Recall. If we now compare that to SSDLite there is a huge difference.

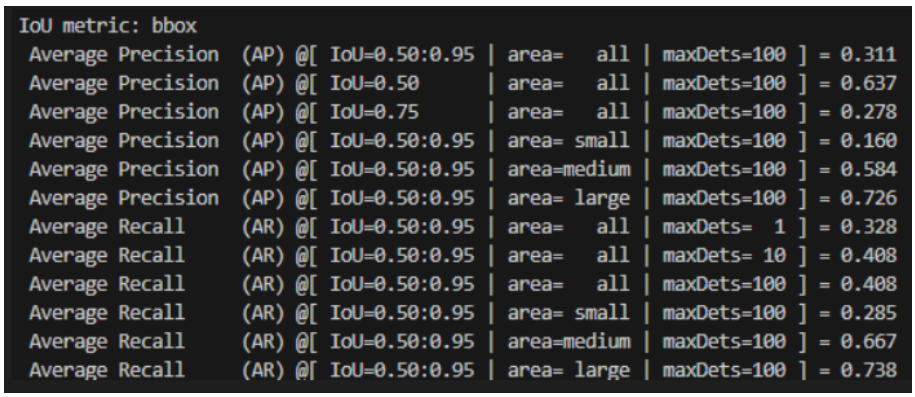

Figure 15: Metrics for SSDLite with MobileNetv3 Large

SSDLite with MobileNetV3 Large after 25 epochs only has an mAP of 63,7% with an IoU of at least 50%. While we can see that the model can recognize medium or large objects with at least 58% in an IoU of 0.50:0.95 the model struggles to recognize small objects. The Average Precision for small objects in the same IoU is only 16%. The performance of this could maybe be improved with a bigger dataset containing smaller licence plates. But as we know there are differences in the architecture of both models. While YOLOv8 can detect all objects really good, SSDLite's architecture is not good enough to capture small objects.

5 Architecture and Metrics for Country Recognition

To achieve real-time in our final project also this model had to be a lightweight model. We used the pre-trained Mobilenetv2. The architecture has a fully convolutional layer with 32 filters and with 19 residual bottleneck layers in addition to that. In the end it utilizes ReLU6 to get non-linearity. The Residual bottleneck layers are components in convolutional neural networks that efficiently extract features by employing bottleneck structures and shortcut connections. These layers reduce computational cost while preserving essential information and enable gradient propagation. They consist of bottlenecks followed by expansions, with shortcuts directly connecting bottlenecks (6).
To adapt the model on eleven classes we modified the classifier by changing the number of the output features of the linear layer.

5.1 Training and Evaluation

As loss function we used the Cross Entropy Loss and as optimizer the Adam Optimizer with a learning rate of 0.001 for the backbone parameters and a learning rate of 0.01 for the classifier parameters.

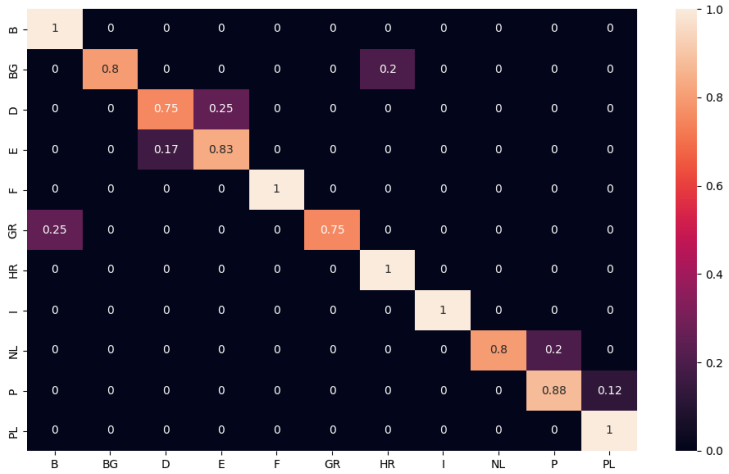

Figure 16: Confusion Matrix for Country Recognition

From the confusion matrix 16 we can see that half of the classes were predicted right while the other classes had a small misprediction. If we look closer we can see that the model struggles to identify countries which have a similar shape like 'D' and 'E' or 'P' and 'PL'. So with that we can clearly recognize that the model needs more data to adapt to smaller differences in the countries.

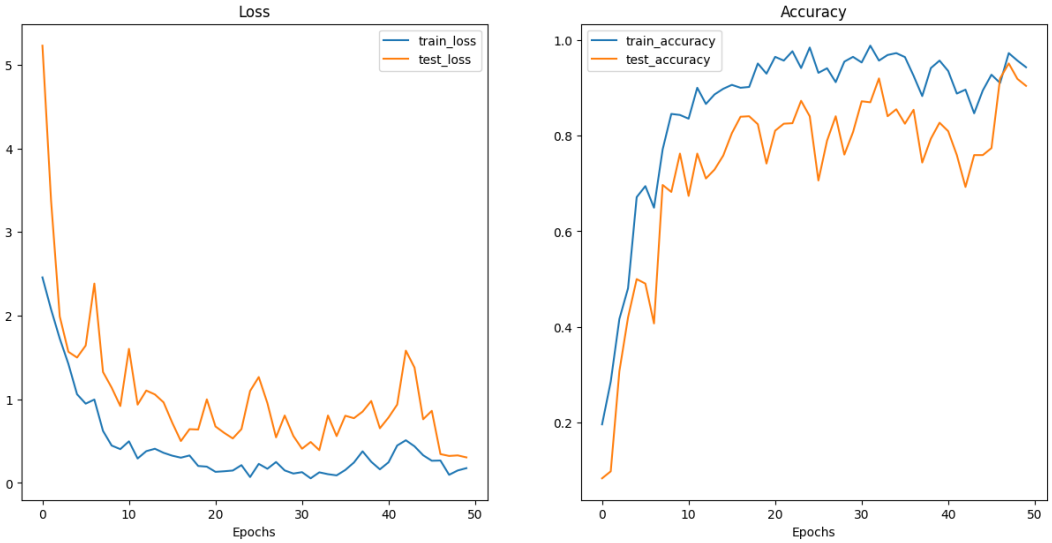

Figure 17: Metrics for Country Recognition

The mAP after 50 epochs is in the train set nearly 93% while in the validation set we get approximately 90% which is, considering the small dataset and testing in the test videos, an actual good performance.

6 Architecture and Metrics for Character Recognition

Due to OCR delivering poor results when the license plate is not clearly visible, we have decided to train a model on a dataset specifically designed for character recognition (4.1.1) instead of using OCR for character detection. This may enhance our accuracy in recognizing characters on license plates in images.

6.1 SSDLite

6.1.1 Training and Evaluation

For training SSDLite, the architecture from section 4.2 is used. We adapted the head to the amount of classes have and used the transformation resizing to 320 and Normalizing with a mean of [0.4,0.4,0.4] and standard deviation of [0.3,0.3,0.3]. We trained SSDLite with MobileNetV3 for 30 epochs with a backbone learning rate of 0.001 and a head learning rate of 0.02. The results of the training are visualized in figure 18.

```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.563
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.788
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.702
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.566
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.525
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.577
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.647
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.647
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.650
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.525
```

Figure 18: Metrics for SSDLite with MobileNetv3 Large

Unfortunately the performance of the model is not sustainably better than using the OCR library. After some testing and checking we found some possible reasons. One reason could be due to the fact of overfitting to the training data, leading to poor performance on new data. But we think that this is not the main reason. The main reason is probably because the dataset contains licence plates of nearly the same size. This leads to recognizing characters only in that or near to that particular size. To illustrate that we let the model through the same 2 licence plates but with a very small variation in size (5 to 10 pixel) and we can see that the model recognizes one perfect and the other one really bad:

Figure 19: Perfect detected image

Figure 20: Bad detected image

Using this model in videos or live cams is not particularly useful for identifying license plates within the footage. But if the task is to find one particular licence plate than this model like the easyocr can be especially useful.
We had the thought to make some transformations in the dataset to support licence plates in other sizes. But as told earlier this dataset also contains images which are too near to the licence plate so making a crop could make the performance even worse.
Another downside of the model is that every time it needs to predict between '0' and 'O' it chooses most of the time 0. But this is because of the poor class balance 11 in the dataset. The dataset has only 36 annotations of 'O' while it has 317 annotations of zero which is nearly 10 times more.

6.2 YOLOv8s

6.2.1 Training and Evaluation

Due to the fact that SSDLite is also delivering poor results in recognizing characters, we decided to train YOLO using the same architecture as explained in section 4.1.1. We trained YOLOv8s for 50 epochs with a learning rate of 0.01 using the dataset explained in section 3.3. Doing so, the results of the training are very good as shown in figures 21 and 22. The only flaw is that the letter 'Q' is not recognized well, mainly because the dataset has too few images for training.

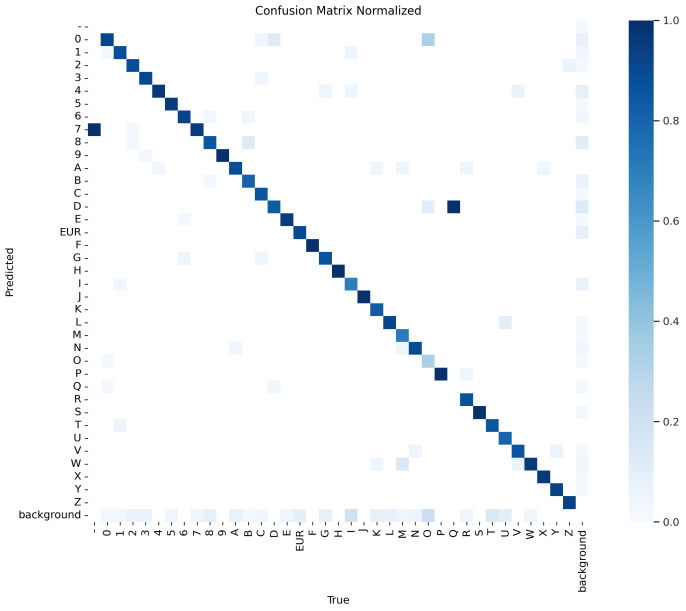

Figure 21: Yolov8s confusion matrix for country recognition

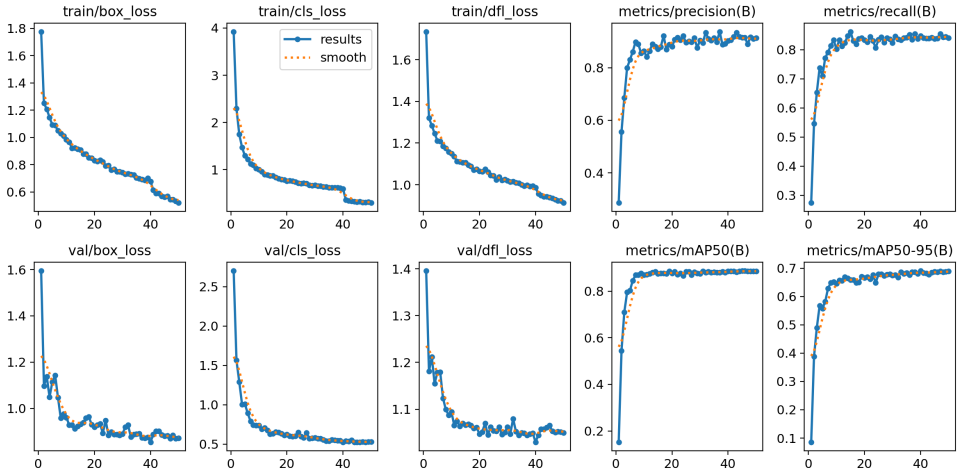

Figure 22: Yolov8s training results

6.3 EASYOCR vs. SSDLite vs. YOLOv8s

To provide an analysis of the effectiveness of the three methodologies for character detection in license plates, the following section presents a comparison of each approach. As shown in the images, using an image even of an good quality, EASYOCR and SSDLite do perform sometimes bad, like in this image:

Figure 23: Test image for three models

In SSDLite we get the following result:

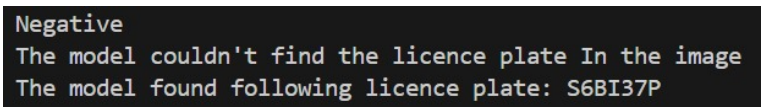

Figure 24: Result SSDLite

In EASYOCR we get the following result:

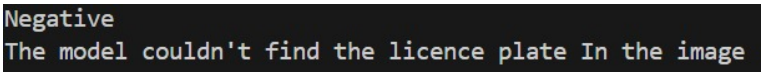

Figure 25: Result EASYOCR

In YOLOv8 we get the following result:

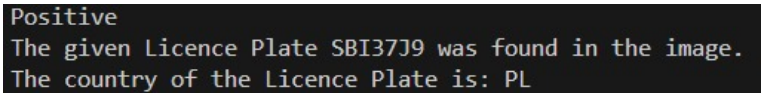

Figure 26: Result YOLOv8

So we get a perfect result in YOLOv8. We also get good results in poor quality images.

7 Website

We also build a Website to make our project possibly remotely available. We will first take a look at the tools we used to build it and after that what features are implemented.

7.1 Flask

For our web application, we used Flask. It is an easy to use micro web framework written in Python. The micro in 'micro web framework' stands for the very simple kept core that allows the developers to add additional libraries and components as needed for the specific project you work on. It is designed to be lightweight, flexible and easy to run, yet the performance may be limited by the use of Flask itself.
We chose Flask, because our models are written in python and that makes it convenient to use a python web framework. Flask uses routes to map URLs to specific Python functions, which allows you to define which code is executed when a certain URL is accessed. You can see this routing where `'@app.route()'` is used to specify the URL and the function below this line of code handles the request.
In the following you can see a an example for the functionality of deleting a video you uploaded, which is one of the features we later get into:

```
@app.route('/delete/<filename>', methods=['GET'])
def delete_video(filename):
    os.remove(os.path.join(app.config['UPLOAD_FOLDER'], filename))
    global videos
    videos = get_existing_videos()
    return redirect(url_for('index'))
```

Here you can see when the URL is for example:

```
https://www.name_of_website/delete/<filename you want to delete>
```

the function `delete_video` gets executed and removes the video from the upload folder, which is a static folder on our server. After deleting the video you get redirected to the index page, which is the landing page basically. The user simply sees the video disappearing from the gallery and does not see the redirection to the index page visually.

Another big convenience of Flask is the handling of Templates. In the passage before we talked about the `index` page, which is one of the templates used in our website. In Flask you can build templates in HTML and insert variables, control structures and macros into the HTML markup as you can see in the following code example:

```
<p>The Number plate {{ numberPlate }} was found on frame {{ frame_number }}.</p>
```

Here you can see the that `numberPlate` and `frame_number` were forwarded from the python function that returned the page that marks up this paragraph in HTML. The return statement in the python function looks something like the following.

```
return render_template('run_model.html', frame_number, number_plate)
```

The first parameter is the template that gets rendered and the next two are variables inside the python function, that also get forwarded and then can get used inside HTML as you can see above in the `<p>`. Another big feature of Flask is the debug mode in which you can change something in the code and you just have to reload the website instead of restarting the whole server.

7.2 Features

Our website has several features which will all be described in this subsection.

You can see our landing page in figure 27. The video gallery is already filled with some example videos.

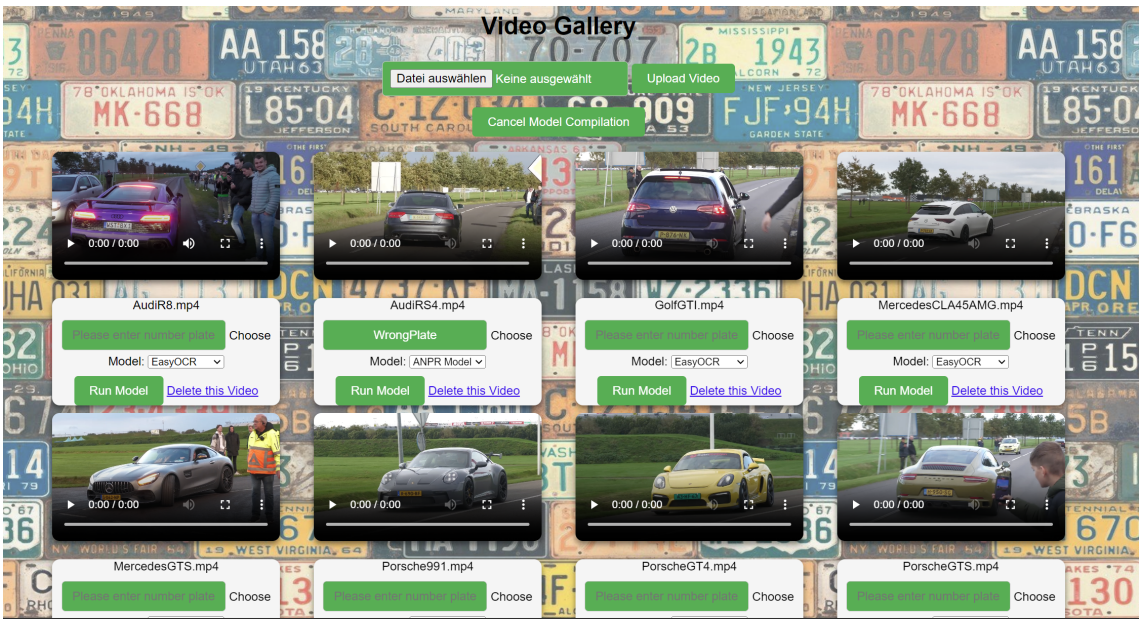

Figure 27: This is the landing page of our website already containing some uploaded videos.

It was important to us to make it as user friendly as possible, so you can choose a video file to upload into the gallery. The video gallery itself then shows the video and its label. You can play/pause the video, watch it in full screen mode and of course change the volume level. There are also options available 'behind' the three dots like the download option (makes not much sense we know...), changing the playback speed and a picture in picture mode. If you hover over each gallery item there is an animation that scales the item in x and y direction by a factor of 1.1 to give the user nice optical feedback. Each item in the gallery has an input text box in which you can type in the number plate you want to search for in the video. Also the user can chose the model which he wants to use for the inference which are 'EasyOCR' and the 'ANPR model'. To correctly get the text of the item in which the `Run Model` Button was clicked, we named the input text box like the following:

```
name="input_numberPlate{{video.filename}}"
```

This is the `name` attribute inside the text box tag. The filename of each video is used to name the input text box, to make it afterwards easy to find. The only problem then would be if you upload a video twice, because then you would have to differentiate between those two in another way. This would be something we would fix in the future. Okay, so now that the user pressed the `Run Model` Button, the inference starts and our server redirects you to the results page. There is also a `Cancel Model Compilation` Button, which does exactly what it's name says. If you press it you get redirected to the landing page and the model compilation get's cancelled.

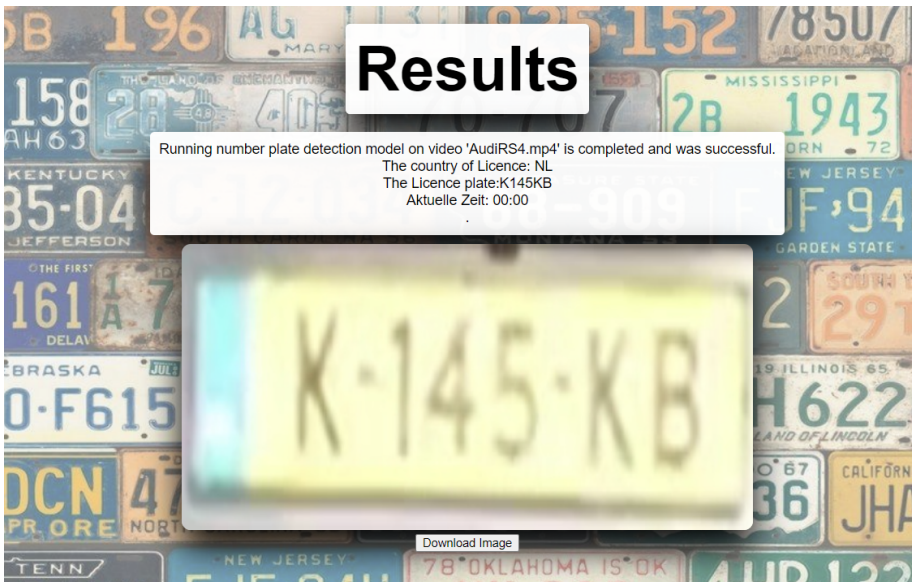

Figure 28: Results page after the model found the input number plate in the video.

In the example of figure 28 the input number plate was found on the frame 0 and the country code was also detected as the one for Netherlands. The frame on which it was detected is also shown on the page and the user has the ability to directly download that picture when clicking on the `Download Image` Button.

If the input number plate could not be found in the video, then you get redirected to a results page in figure 29. Here there is a funny little GIF of the famous scene in Pulp Fiction, where John Travolta is looking confused from side to side as if he is looking for the number plate. On the bottom there are a few tips for the user like making sure the spelling is correct and the number plate is in the video.

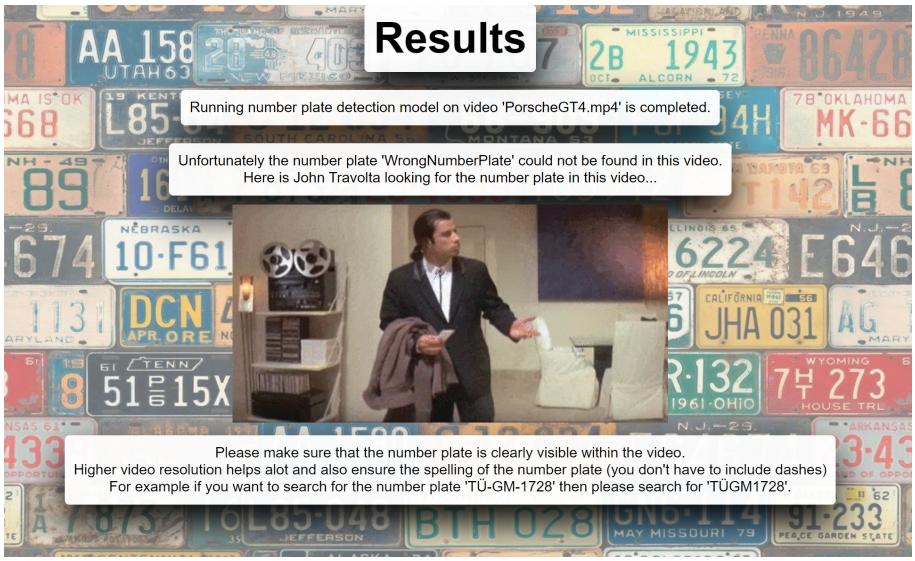

Figure 29: Results page if the input number plate was not found in the video.

8 Conclusion (and Future Improvements?)

In summary, the outcomes of our project suggest a successful implementation in detecting and reading

license plates and also determining their country of origin. This success is particularly notable following the shift from optical character recognition to a model explicitly trained for character detection. This enhancement has improved the system's accuracy and efficiency by a lot. It is now capable of reading characters in images of bad quality and is not relying on getting a license plate in high resolution, good contrast and not tilted.

8.1 More training / bigger dataset

The primary enhancement we could pursue involves extending the training duration of our models while utilizing higher quality datasets. Often, our training sessions were constrained by resource availability, necessitating the use of a private GPU (RTX 3070 Ti) due to continuous usage of all GPUs in the cluster. Additionally, our datasets were somewhat limited, in terms of representation from a subset of all the European countries, and the available data within each category was insufficient. As depicted in Figure 11, another issue we encountered was the significant class imbalance, exemplified by instances such as the letter 'O' having only 36 images, whereas the number 3 had a considerably larger set of 228 images. To address these shortcomings, allocating more resources for longer training periods on superior datasets would likely yield more robust and accurate models. Moreover, diversifying the dataset to include a broader range of countries and ensuring a balanced distribution of samples across classes would contribute to improving the model's generalization and performance.

8.2 Integration of superresolution

One aspect we didn't address but could potentially explore for future enhancements involves employing super resolution techniques to augment image resolution. This could prove particularly valuable for scenarios such as security cameras operating at lower resolutions or instances where capturing license plate details from a distance results in suboptimal resolution. We already did a little research which indicates the availability of deep neural networks that seamlessly integrate with OpenCV, presenting promising avenues for advancing image resolution enhancement. Incorporating such capabilities could significantly enhance the accuracy and effectiveness of our system, especially in scenarios where image clarity is compromised.

8.3 Enhancing/ improving user experience

In terms of usability enhancements for future iterations, one potential avenue could involve the development of a mobile application. Offering a mobile app would cater to user convenience, as accessing our service via smartphones has become increasingly prevalent and desirable. While our website remains a viable platform, the demand for a dedicated app could arise if enough people would use it. Moreover, there's room for further refinement of our website's functionality. For instance, implementing a feature that retrieves all number plates along with their corresponding country codes and timestamps from selected videos could prove invaluable, especially for applications like parking spot monitoring bots, facilitating efficient tracking of parked vehicles. Additionally, incorporating user-centric improvements such as personalized accounts for accessing saved files or even fostering a community platform where users can share pertinent information, such as images of vehicles involved in theft incidents (subject to legal considerations), could enhance user engagement and utility.

Another significant enhancement pertains to the infrastructure side, particularly ensuring that the server is equipped with GPU capabilities, potentially multiple GPUs based on demand. Access to GPU acceleration substantially improves the user experience, as our models operated approximately 100 times slower on CPUs compared to GPUs like the RTX 3070 Ti we used. This enhancement not only improves processing speed but also enables real time processing of videos, contributing to a smoother and more responsive user experience overall.

References

[1] U. of Southern Philippines. (2024) license-plate-yu5q-license-plate. [Online]. Available: https://universe.roboflow.com/university-of-southeastern-philippines-cnl9c/license-plate-detection-merged-projects/dataset/3

[2] jML, "lp-detailed dataset," https://universe.roboflow.com/jml/lp-detailed, aug 2022, visited on 2024-03-14. [Online]. Available: https://universe.roboflow.com/jml/lp-detailed

[3] A. Mehra. (2023) Evolution of yolo object detection model from v5 to v8. [Online]. Available: https://www.labellerr.com/blog/evolution-of-yolo-object-detection-model-from-v5-to-v8/

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, *SSD: Single Shot MultiBox Detector*. Springer International Publishing, 2016, p. 21–37. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46448-0_2

[5] V. Vryniotis. (2021) Everything you need to know about torchvision's ssdlite implementation. [Online]. Available: https://pytorch.org/blog/torchvision-ssdlite-implementation/

[6] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," 2019.