# 1. Dataset

## 1.1 Explanation of the dataset

Our dataset consists of various vehicles of different brands. Since our group only consists of two members, we decided to keep the number of brands that our model will be able to detect to a minimum. We chose to detect five most common brands that are found in the Netherlands, those being: Audi, Mercedes, BMW, Toyota and Volkswagen.

Due to the fact that the hardest challenge for us was to find an appropriate dataset, mainly because there are a lot of images and videos that are either very blurry or not in traffic condition, we decided to collect around 125 images per class. We wanted to provide slightly more images than mentioned in the requirements, and this is for each of our classes. This means that in total, our cleaned dataset consists of 600+ unique images spread over five categories. The dataset was split up into 70% training, 20% validation and 10% test. After the appliance of augmentation of the data, it got split into 88% training, 8% validation and 4% test data.

The gathering of the images was done through frame-extraction using Python libraries such as CV2. The source for these images were taken from various YouTube clips in which the five specific brands were clearly visible within the frame. We did not want to keep it like this because we wanted the vehicles to be in traffic condition on the roads, rather than relying on various stock images available on the internet. We had encountered various stock images datasets and they did not contain variance nor were they used in any traffic condition. Due to this, we considered them as artefacts to be excluded.

The annotation process was done by drawing a bounding box around the logo objects on the cars and classifying them accordingly by their corresponding brand. The annotation itself was done in Roboflow which was relatively straightforward as it provides you with the right tools and guidance.

On top of this, we also augmented our data by providing techniques such as horizontal and vertical flipping, 45-degree rotations, grayscale and hue. The main purpose behind this decision was due to the fact that we have a limited dataset and we wanted to increase our image size to essentially increase the chance of getting higher accuracy on our model.

The choice behind the specific types of augmentation techniques reflects our vision of how we want the model to function. Namely, the logo should be detected under any angles, flips and any types of coloring. After all, we want our model to determine the brands by the shapes of the logos and not their color or size or their positioning.

## 1.2 Individual contribution

### 1.2.1 Alexander Arkhipov – 647833

The separation of tasks between my teammate and I was relatively straightforward. We agreed that I would take on the extract of frames from three of the brands, which are: Volkswagen, Mercedes and BMW. I was thus responsible for gathering 125 images in each three of these categories from various YouTube video clips. The collection of the images for the dataset did not prove to be a hard task, but a rather slow one because not all videos have appropriate brands, clips or even logos on the cars of interest. On top of this, I was also responsible for annotating these images within Roboflow. The annotation process was done in Roboflow as they provided the rights tools and made the whole process of drawing a bounding box and classifying the object a light task. The augmentation appliance was relatively easy since Roboflow provides that for you, but I was in fact the one who chose to go with the designated effects based on a recommendation from our Computer Vision teacher as well as previous experience when applying them previous datasets.

### 1.2.2 Mathijs Kroese - 655419

When dividing the tasks between us, I focused on collecting images of Audi and Toyota for our dataset through the extraction of frames from various YouTube videos. I was responsible for collecting around 125 images of both classes. The collection of enough images for Toyota proved difficult to do from YouTube videos however, as there were rather few videos containing frames where the Toyota logo was visible in such a way that it could be useful for our dataset. Following this, we decided to go for the next best thing, and look for images of second-hand Toyota cars on AutoScout24. This allowed us to gather enough images in which the Toyota logo was visible from different angles. After collecting the images, I annotated them using Roboflow and added them to the dataset to be used for training our model.

# 2. Model

## 2.1 Explanation of the model

For our definitive model we chose to go with YoloV5. We have tried two Yolo models which are V5 and V8. We have also tested both of these models under various different settings such as number of epochs, batch-size and optimizers under which we receive different results. These settings and results are further evaluated in the individual contribution parts. In terms of the choice behind Yolo specifically, the bottom portion explains what we have tested and research and why we came to a conclusion as to why Yolo is the appropriate model in our situation.

We chose these two models because they reflect our use case for the project. We are in charge of developing a deep learning algorithm which can detect five different brands based on their logo. With this in mind, we concluded that this is an object detection task for many reasons. Firstly, we are dealing with various objects on the road for which we have to specifically detect the logos of the vehicles. Although this might seem like a multi-class classification task at first, we cannot perform this without object detection. In essence, we have to be able to detect multiple brands in one frame or video which means that simple classification will not suit the job. This is rather a multi-class object detection task in which there are multi-labels. Yolo essentially allows us to predict multiple bounding boxes per frame which is extremely useful if more than one car is within the frame.

Prior to deciding to use Yolo, we initially started to do research on other object detection algorithms such as Faster R-CNN and SSD. When it comes to Faster R-CNN, we concluded, although it is very good for accuracy due to the fact that it is a two-shot-detector and it scans the whole architecture twice, it is still very slow in real-time application. In other words, we would not be able to use this in real-time scenario which is bad because our use case can very well be applied in real-time, such as surveillance cameras. Faster R-CNN also proved to be quite difficult to implement as there were a lot of components which were complex to understand, and it required many more lines of code to get the whole process going. We also looked at the possibility of the use of the Single Shot Detector (SSD) model. When it comes to SSD, the model has its positives and negatives. The SSD model has shown impressive accuracy, but this accuracy comes at the cost of a reduced inference speed, making it less suitable for real-time traffic videos. Another point of consideration when looking at the SSD model was the ease of use and the training of the model itself. The complexity involved with SSD proved it difficult to implement and finetune, while having many parameters, which results in a large size model that takes longer to train. Lastly, the accuracy of SSD models is highly reliant on large datasets. Considering the dataset that we used to train our models consisted of around 125 images for each class, the SSD model would not be fully optimized.However, when we looked at Yolo, we noticed that it had a good balance of speed and accuracy, which are both essential components when it comes to the detection of brands in traffic environments, especially in real-time. On top of this, it also offers an extremely easy way to perform training and detection processes by just using a few CLI commands. With this, it also allows you to easily fine-tune the model by providing various hyperparameters in the same line of code, such as epochs, image-size, optimizers, batch-size, etc.

After deciding on the model, we still had to figure out which version of the Yolo model we were going to use and apply for our dataset. We thus set out to test different Yolo versions under which we tested YoloV5 and YoloV8. By going through various training and testing sessions of both models, we concluded that YoloV5 would be more suitable since it does not only provide ease of use, but the training process is much faster than YoloV8, at least in our case. Furthermore, YoloV8 is still under heavy development, so we also thought it would be more appropriate to go with a more stable version, resulting in our choice of YoloV5.

## 2.2 Individual contribution

### 2.2.1 Alexander Arkhipov – 647833

I took it upon myself to test the YoloV5 model on our dataset. To achieve this task, I have followed various tutorials that are available on YouTube and looked though many GitHub repositories to gain insight on how the model works and operates. I eventually stumbled upon a tutorial which was suited for my use case. I started off by importing various libraries which I would need for training the model. Since YoloV5 uses PyTorch, I had to read a brief documentation on how that works. However, since YoloV5 uses CLI commands, it made the training process much easier for me to perform and understand.

I first started off by downloading the cleaned and augmented dataset from Roboflow and cloning it onto Google Collab. For this training process, I used the YoloV5 pretrained model, but I did not apply any pretrained weights. This means, no transfer learning was applied to the weights that I got after the training process. The hypermeters that were provided to this model during the training process were the number of iterations (epochs), batch-size and the optimizers. I have performed a total of four tests, where I used different number of epochs, batch-size and switched between two optimizers.

My first training session started off with 100 epochs with a batch-size of 8 and an optimizer of SGD (Stochastic Gradient Descent). The accuracy delivered by this was 88%. However, this was not optimal because when looking at the precision, recall and mAP graphs, I noticed that there is a lot of room for further training as it did not reach the maximum potential. In other words, the line kept going up without reaching a clear stalemate (where it goes horizontally flat which indicates maximum potential of the model).

For the second training session, I bumped up the epochs to 300, switched the batch-size to 16 and left the optimizer on SGD. The accuracy now was 95%, which was a significant improvement on last time. Despite the good result, there was still a case of overfitting. After looking at the graph, there was a flat horizontal line ranging from 200-300 epochs. Due to this I ran a third test where I kept everything the same and only changed the epoch size to 200. In the end, it ended up yielding nearly 95% accuracy as well, but it decreased the training time tremendously.

For my fourth test, I decided to keep everything as it is (200 epochs, 16 batch-size), but this time, I changed my optimizer to Adam to see how the model would perform on an adaptive learning algorithm. The results of this training session were surprisingly better, the model reached an average accuracy of around 96%, which is a 1% increase from last time. The average loss of this last training session was 0.025686 where box loss was 0.01792, object loss was 0.005704 and class loss was 0.002062.

The extra graphs of the mAP, recall, precision, confusion matrix and more are available within the notebook which I made for the teachers to test the model using my weights. The training of the model can be performed using the link to Google Collab which I provided in the notebook. Furthermore, it also contains all of these graphs which can easily be viewed for further analysis. On top of this, I also provided a way to inference the weights gained for the training session, using the pretrained YoloV5 model on new test images. All of this can also be found in the notebook provided.

### 2.2.2 Mathijs Kroese - 655419

After we decided to go for the Yolo model, we decided to look into the performance of the Yolov8 model with our dataset, as well as the YoloV5 model, so that the results could be compared, and the most optimal model could be selected.

The first steps consisted of looking up guides and information surrounding the use and workings of the model. During this, I found a link to a Roboflow Google collab, where they provide a tutorial on how to install Yolov8, provide an example of the results with the model, and finally explain how to train a model using a custom dataset generated through Roboflow. When following this guide, I discussed with my teammate who was working on the YoloV5 which image resolution and how many epochs were used during the training of that model, so that the configuration of the Yolov8 mirrored those settings. The goal was to compare the results of YoloV5 and Yolov8 with as few variables as possible. While training the Yolov8 model however, the training process took longer than anticipated, and after roughly 3,5 hours, the Google collab file stopped the training process as I have used up the amount of available GPU power that was available to me as a free user. At this point, the model was just past halfway through the epochs, implying that the total process of training the Yolov8 model would take roughly 6 hours. Seeing as we wanted to try out different settings to compare the results, we agreed that the training process was not feasible, thus ruling out the Yolov8 model and deciding to go with the Yolov5 model, as the training was many times faster.

Following this conclusion, the next step taken was to look at the results from the training and testing of various YoloV5 models by my teammate. In the time it took to try and train the YoloV8 model, my teammate was able to train two Yolov5 models: one model trained with 100 epochs and a batch size of 8, and one model using 300 epochs and a batch size of 16. Seeing as the results showed that the latter model scored higher on the accuracy, with a score of 95% compared to 88%, we looked at the data and noticed that around the 200th epoch, the graphs drawn to represent the model’s performance, the line seemed to flatten. Based on this we decided to take two routes: we both would train a YoloV5 model using 200 epochs and a batch size of 16, but my teammate would change the optimizer to Adam, whereas I kept the optimizer on SGD. By comparing the results, we found that the Adam optimizer performed slightly better with an accuracy score of 96% compared to the 95% achieved by the SGD optimizer.

# 3. TFGD

## 3.1 Explanation of TFGD

The TFGD of the application consists of various parts. The first part consists of the function design of the proof-of-concept application. This portion consists of use case and activity diagrams which represent the flow of the application. The second part is made up of the technical design for the proof-of-concept application. This part is defined with the sequence and the class diagrams which represent the technical structure as well as the flow and interactions between various objects and components of the application. The third part consists of the proposed proof-of-concept graphical design for which a collection of wireframes and an interactive prototype has been delivered to test the functional design of the application within the Figma environment.

## 3.2 Individual contribution

### 3.2.1 Alexander Arkhipov – 647833

For the TFGD, I participated in the making of both the functional and technical design of the application.

For the functional design I participated in the making of the use case and activity diagrams. I primarily focused on our use case for the brand detection, but I also gave tips and my insights to the rest of the group that was doing the license plates detection as we ended up merging the TFGD of both the brand detection and license plate detection models together. Thus, it was important that we worked together and understood what each member of the group wanted.

For the technical design, I participated in the making of the sequence and class diagrams. As previously stated, I focused more on the brand detection part for both of these diagrams, but I also gave my feedback to the other group as we merged everything together.

I was also responsible for providing a small description above each diagram to give an insight of what the diagrams represent and what they are used for. On top of this, I also mentioned what the several components do and gave a brief overview of the overall flow.

I did not participate in the making of the graphical design as another member of the other group took it upon himself to deliver that portion of the TFGD since he was working for it from the start.

### 3.2.2 Mathijs Kroese - 655419

I worked together with the rest of the team on the TFGD to create the functional and technical design. For the functional design, I participated in the creation of the use case diagram, sequence diagram, and the activity diagram. As a team, we set up a meeting via Teams to allow us to work together on the diagrams. My participation consisted of giving my insights, helped make the class diagram, provided feedback, and edited parts which could use improvement.