Efficient Traffic sign and Object detection implementation for a Self Driving Car

D.H. Honderd

***Abstract*—Autonomous vechiles are revolutionizing transportation, yet their efficacy depends on sophisticated object detection systems capable of real-time performance under resource constraints. This thesis delves into the creation and optimization of a object detection system for a self-driving car, leveraging the YOLO-v5 framework to identify and classify traffic signs, pedestrians, and other vechiles. Through the use of a combination of quantization and downsampling, the system achieves high processing speed and accuracy, while running on a modest CPU computational power of an Intel NUC mini-computer.**

This thesis shows the application of the RDW Self-Driving Challenge 2024 in which this system was used for the University of Twente team. The results highlight third place in the competition and a system that is capable of stopping before red lights, adhering to the speed limit, letting a pedestrian cross and spotting cars.

The discussion highlights the trade-offs between accuracy and processing speeds, the possible impact of different optimization techniques, and the practical considerations and adaptations it would need for the real world environment. The findings provide a valuable groundwork for future teams of the RDW challenge and overall object detection for autonomous vechiles.

I. Introduction

Self-driving cars, also known as autonomous vehicles, have been a topic of interest and development for decades. The concept of a self-driving car was first introduced in 1939 during the World's Fair by General Motors, envisioning a future where cars could drive themselves [Mot39]. A major part of self-driving cars consists of interacting with traffic signs and acting accordingly. This requires algorithms or models that look through the camera of the self-driving car and can recognize and classify the different traffic signs and objects.

The importance of accurate traffic sign recognition is underscored in competitive events such as the RDW Self-Driving Challenge (SDC) organized by the RDW. This annual event, starting from 2019, offers a platform for young talent to develop and demonstrate their skills in smart mobility. The 2024 edition of the challenge will take place at the RDW Test Center in Lelystad, where student teams will navigate a custom track designed to mimic real-world driving scenarios [RDW23]. These scenarios include recognizing and reacting to traffic lights, complying with speed limits, stopping for pedestrians, overtaking stationary obstacles, and performing complex maneuvers, such as parallel parking.

For this thesis part of the work will be subjected towards this challenge for 2024 consisting of the real-time subsystem for traffic object detection and recognition. Not only will this thesis work on a system that can perform object detection on speed limits, pedestrians and cars. It will also develop a system that can make decisions based on the information that it detects.

A. Background and Motivation

The most common way of object detection is through convolutional neural networks (CNNs) and frameworks like YOLO (You Only Look Once) further revolutionized the field. The YOLO algorithm, with its real-time object detection capabilities, has been widely adopted for traffic sign recognition in advanced driver-assistance systems (ADAS) [Nam22]. Optimizations such as pruning and quantization have made these models more efficient, enabling deployment on low-power edge devices like FPGAs (Field Programmable Gate Arrays) or implementations with only a CPU and no GPU as is the case with the self driving car.

However, there are a few problems with all the work that has already been done like an object detection system with YOLO, most of it being company assignments, resulting in that none of the code or documentation of this progress mentioned is open source. This makes it difficult to use these previous studies besides their methods, techniques and literature to use as a foundation to build off of.

Second, training a model like YOLO requires a dataset to train it. As will be discussed later, there are currently datasets like mapillary or GTSRB, which contain most of the common traffic signs such as all the speed limits, stop signs and much more. However, the collection of classes that need to be detected in this scenario are not part of any public dataset like the 10 km/h and 15 km/h speed signs. This requires the creation of a custom dataset tailored to the requirements. [Map24] [Ins24] [Ert+20]

It is important to know beforehand of creating an object detection system the specifications of the system that it will run on. The relevant specifics to object detection in this self-driving car system are:

- Intel NUC mini-computer
- Intel® CoreTM i5-1135G7 processor with 16 GB RAM and 4 cores and 8 threads
- 3 x Logitech StreamCam (1920x1080 @ 60 fps, 78° FOV, 540 focal length), however these are set to 848x480 and 30 FPS to reduce the system load, and only the middle camera will be used for object detection

[RDW23]

A system that will implement object detection and only contains a decent CPU requires it to be implemented efficiently to not overload the CPU, because as it is running on the self-driving car, also other subsystems will run, like steering to stay between the white lines and communication between the car and the NUC.

B. Requirements for the Self-Driving Car

For a self-driving car to operate safely and efficiently, it must meet several key requirements:

- Accurate detection and classification of traffic signs, traffic lights, pedestrians and cars, can detect objects within required distance thresholds.
- Real-time processing capabilities to interpret objects quickly. That means stopping before a red light, adhering to the speed limit, allowing a pedestrian to cross, and detecting a car.
- Distance estimation to the detected objects to react to them in time
- Robustness to varying lighting conditions, weather, and occlusions.
- Integration with other autonomous vehicle systems for decision making and navigation.

II. Related Work

A lot of work has been done in the field of object detection mostly with YOLO models, dataset creation, distance estimation from images and integration of real time decision systems.

A. Object Detection through YOLO models

There are a lot of different algorithms that can do object detection in this scenario. The most popular options are YOLO, R-CNN(Regional Convolutional Neural Network) and its variants and SSD(Single shot detector). These different type of models are discussed in [Jia+19]. Where I concluded from that R-CNN are overkill for this scenario and would most likely overload the CPU and that SSD is better at small and overlapping object detection but is generally slower and more complex to implement and train then YOLO, leaving YOLO as the best option for the real-time requirements. Within the YOLO family there are a lot of different variants, in a study published July 2023, multiple different versions of the YOLO models were tested, specifically for the use of traffic sign recognition. What was concluded from this study is that YOLO-v5 was the best version for this task [**yolo˙systematic˙review**]. The ultralytics open-source project of YOLO-v5 is one of the best frameworks for building a object detection system [Ult24b]. The frameworks comes with model trainers, pre-trained-models to build of from, evaluators and automatic optimization techniques like pruning, it allowed a very quick creation of a model.

The YOLO algorithm can be quite complicated to fully understand, but it is not worth mentioning the intricacies in this paper; for a detailed explanation of the structure, refer to papers [Ami+23] and [**yolo˙systematic˙review**].

Training a model through the YOLO framework is made as easy as possible [Ult24b], besides needing a dataset to train a model, the optimal training parameters and hyper parameters also need to be determined, this paper used for detection of pest has a section dedicated for optimal training parameters and I used it as a starting ground to start my base parameters from. [Ami+23]

B. Dataset creation

Creating a dataset for a convolutional neural network has been done before, a detailed tutorial of how it works is explained in [Ult24a]. In addition, a key part of creating a dataset is balancing the data sets class occurrences to prevent bias inside a dataset. This was done inside the study of [Sin20]

C. Distance estimation

Distance estimation from monocular images is essential for autonomous driving and has been extensively researched. Previous studies, such as those of Faculty of Applied Sciences, Macao Polytechnic University, utilize the known object dimensions and camera parameters to accurately compute distances. This is done through leveraging the camera's focal length and the objects real-world dimensions to determine distance, which is crucial for the self driving cars to react timely to their environments. For the formulas used please refer to subsection III-B Distance estimation. [Cho+23]

III. Methodology

The methodology of this thesis is divided into several key stages, each with a specific task and goal aimed at requirements of the detection system for a self-driving car. This section will cover dataset creation, distance estimation, model training, evaluation metrics, integration, and optimization strategies. For all of the code of this project refer to the appendix section VI-A.

A. Data set

A dataset is used to train a CNN model, it consists of labelled images. To obtain the proper accuracy of the model, it requires that the data set consists of a wide variety of circumstances in its classes. Such as different weather conditions/ lighting, different angles and distances from the classes and different image augmentations such as zooming, blur, contrast/ brightness and flipping. After that to prevent bias for certain classes, the dataset needs to re-balanced if certain classes are under-represented.

The creation of the data set can be summarized by the diagram in Figure 1.

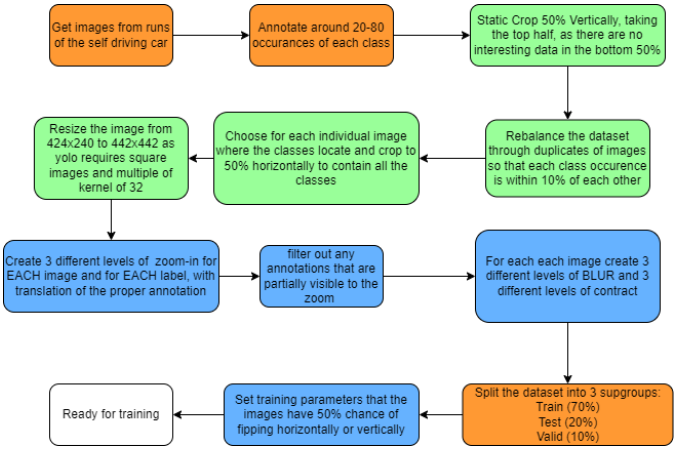

Fig. 1: The flow chart of the database creation with small motivation for each step, the blocks are coloured to show the specific type of step it is:
Green: Image preparation,
Blue: Image augmentation,
Red: Overall dataset creation and preparation

1) Class definitions: The object detection on the self driving car needs to be able to detect the following classes:

- Traffic Speed sign 20-km/h, width: 0.6m, height: 0.6m
- Traffic Speed sign 15-km/h, width: 0.6m, height: 0.6m
- Traffic Speed sign 10-km/h, width: 0.6m, height: 0.6m
- Traffic Light Green, width: 0.07m, height: 0.3m
- Traffic Light Red, width: 0.07m, height: 0.3m
- Pedestrian, width: 0.5m, height: 2m
- Car, width: 1.8m, height: 1.4m

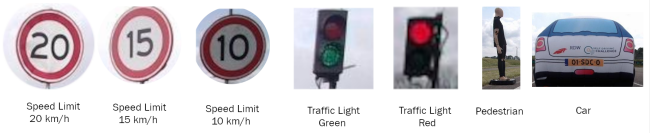

Fig. 2: The object classes listed in the class definitions

The visual representation of these classes can be seen in Figure 2.

2) Collecting live images from the track: The first step in creating a new data set is to have a database of images containing the classes mentioned above so to achieve this, live video data from the self-driving car throughout the track was recorded, this included the same objects for different weather and lighting conditions. The next step was to select certain frames that contain the classes to add to the data set. It was done from the perspective of the car to assure that once the object detection runs on the car itself, there would be no change in perspective compared to its training.

3) Annotation of the images: Before an image is ready to be used for training, its first annotated by hand, so that the model knows where in an image the object is. This is done with the help of tools such as Roboflow. The process, explained simply, consists of going through an image, finding the area of interest, and labeling it as one of the according predefined classes.

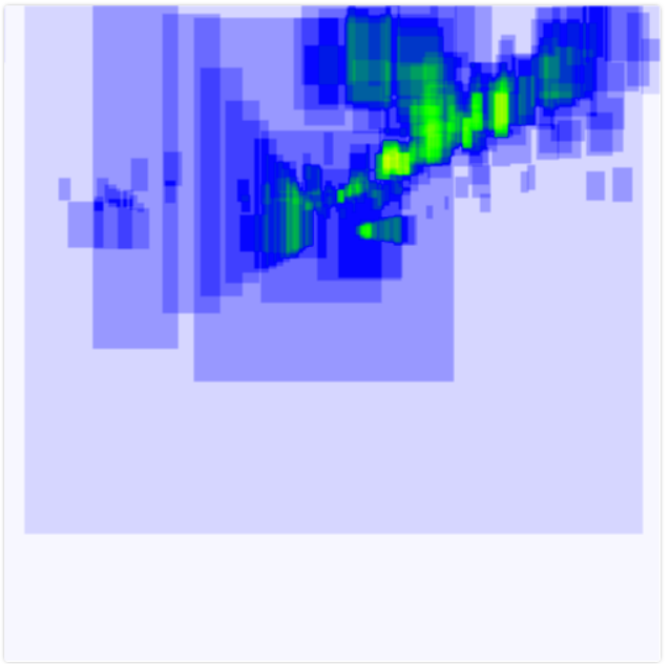

Fig. 3: In this picture a heat-map can be seen of where all the annotations reside, where it can be seen that there are no relevant annotations in the bottom 50 % of the image, besides a picture containing the whole car

4) Image preparation: After all annotations were completed, it was noticed that in the heat-map Figure 3 that in the bottom 50 % of the images, there was no relevant information, and thus not relevant for training. After that because the YOLOv5 framework works only with square images, and that most detections can be found within a 50 % width of image width of the image, the dataset could be optimized to further reduce noise, by cropping the images to only include the 50 % of the image width that contains the classes, and if not all classes fit within that just split it into 2 separate images in the training data. The region that is included in the crop is based on where the annotations are located.

5) Data Augmentation: Data augmentation was done at 2 different levels, the yolo-v5 framework before training has some built in image augmentation that can be set in its training configuration file, but not enough to solely rely on it. Thus to fill out this augmentations they are listed below:

- Zoom: different levels of zoom of the same class, which means that the images in the dataset will be trained on different surroundings of the class its trying to detect, it also allows more detailed
- Blur: Different levels blur, to be robust enough that even if there is motion blur or other types of blur it could still detect the image.
- Contrast: Besides the different real world lighting conditions, create artificial levels with increased and decreased

lighting

These augmentations assure that the model is trained on different scenarios of lighting, blur and further distances from the car itself.

The augmentations that are done through the YOLO-v5 framework:

- Rotation: Rotate the images slightly (up to 5 degrees)
- Horizontal flipping, basically mirroring the images

B. Distance Estimation

Distance estimation can be done from monotone images as long as the camera parameters are known and the real metrics of the object that the is distance is being measured from.

As mentioned section II-C the most simple way of object detection is through the use of focal length. In knowing the real width and height of an object, the focal length of the camera and the pixel width and height of the detection, distance can be calculated.

The distance D to the object can be calculated using both the width and height of an object as shown in equations 1 - 3:

$$D_{\text{w}} = \frac{W \cdot f}{w} \tag{1}$$

$$D_{\text{h}} = \frac{H \cdot f}{h} \tag{2}$$

$$D = \frac{D_{\text{w}} + D_{\text{h}}}{2} \tag{3}$$

where: (in meters)

- D_{w} is the distance calculated using the object's width
- D_{h} is the distance calculated using the object's height
- D The eventual distance calculated using width and height
- W is the real width of the object
- H is the real height of the object
- f is the focal length of the camera, 540 in this case
- w is the width of the object in the image (in pixels)
- h is the height of the object in the image (in pixels)

[Cho+23]

C. Model training

Throughout progressing of the RDW challenge and thesis, different models were created, throughout experimentation of the hyper-parameters and training parameters and related work the final hyper parameters can be seen in Table I.

The model was trained using the utwente jupyter servers [Twe24], besides its hyper parameters, it also has training parameters which can be found in Table II.

D. Evaluation Metrics

For this system, there are different metrics to take into consideration to define a real-time robust and efficient system.

Parameter	Value	Description
`lr0`	0.01	Initial learning rate
`lrf`	0.1	Final OneCycleLR learning rate
`momentum`	0.937	SGD momentum/Adam beta1
`weight_decay`	0.0005	Optimizer weight decay
`warmup_epochs`	3.0	Warmup epochs
`warmup_momentum`	0.8	Warmup initial momentum
`warmup_bias_lr`	0.1	Warmup initial bias learning rate
`box`	0.05	Box loss gain
`cls`	0.5	Classification loss gain
`cls_pw`	1.0	Classification BCELoss positive weight
`obj`	0.7	Object loss gain
`obj_pw`	1.0	Object BCELoss positive weight
`iou_t`	0.20	IoU training threshold
`anchor_t`	4.0	Anchor-multiple threshold
`anchors`	3	Anchors per output layer
`fl_gamma`	0.0	Focal loss gamma
`hsv_h`	0.015	Image HSV-Hue augmentation
`hsv_s`	0.7	Image HSV-Saturation augmentation
`hsv_v`	0.4	Image HSV-Value augmentation
`degrees`	5.0	Image rotation (+/- deg)
`translate`	0.1	Image translation (+/- fraction)
`scale`	0.5	Image scale (+/- gain)
`shear`	0.0	Image shear (+/- deg)
`perspective`	0.0	Image perspective (+/- fraction)
`flipud`	0.0	Image flip up-down probability
`fliplr`	0.5	Image flip left-right probability
`mosaic`	1.0	Image mosaic probability
`mixup`	0.0	Image mixup probability
`copy_paste`	0.0	Segment copy-paste probability

TABLE I: Hyperparameters for YOLOv5 Training

Parameter	Value
Image Size	448
Batch Size	3
Epochs	100
Model Configuration	`./models/yolov5s.yaml`
Initial Weights	`yolov5s.pt`
Cache	Enabled
Patience	10

TABLE II: Relevant Training Parameters for YOLOv5 Model

1) Object Detection Accuracy Metrics: There are several metrics that express the accuracy of a convolutional neural network. It should be kept in mind that these metrics are calculated within a validation set of the database which differs on the set that it trains on, which is a useful tool to get an initial estimation of the accuracy, but cannot fully represent how it would perform in real-time in the real world. Some of these metrics use the IOU (Intersection over Union, where the calculations off are shown in equation 4, it shows the overlap between the detection and the original annotation. The different accuracy metrics are shown below:

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}} \tag{4}$$

- **Recall** is the ratio between the true positive detections to the total number of actual positives (true positives + false negatives).
- **Precision** is the ratio of true positive detections in regards to the total predicted number of positive detections (true positives + false positives).

- $\boldsymbol{mAP_{0.5}}$ stands for Mean Average Precision at IoU threshold of 0.5, which quantifies the object detection inside the images.
- $\boldsymbol{mAP_{0.5:0.95}}$ is the Mean Average Precision averaged over the IoU thresholds from 0.5 to 0.95, which means that the object detection is really close to the training annotation.

2) Object detection speed metrics: These metrics represent the speed and efficiency of the system, used to determine how memory, CPU efficient the system is.

- **FPS**: The amount of frames per second the object detection process calculated through the average of $\mathbf{1}$ over the processing time per frame.
- **CPU usage**: CPU usage will be presented as a $\%$ of the potential total CPU usage of the CPU of the NUC.
- **RAM memory usage**: this shows the RAM usage of the Object detection in GB, used to see how memory intensive the program can be.

E. Integration

The integration consists of 5 main parts: initialization, frame acquisition and preprocessing, Inference, Post-Processing and Distance Estimation and Detection and State Management. Each part plays a crucial role to go from a frame captured from an image to instructions for the cars throttle and breaks.

For a detailed overview of how this system operates refer to Figure 4.

1) Initialization: The model is first initialized through the use of the YOLO-v5 framework which is used to handle a variety of models. The model then goes through a warm-up which optimizes the accuracy and processing speed. After the model will go through quantization, originally the parameters of the model each consist of 32 bits, this can be lowered to 8 bits, which can reduces the calculation size for each detection. In the section III-F an experiment is run to test different optimization combinations between quantization and downsampling.

2) Frame acquisition and Preprocessing: Frames are captured from the front facing camera of the car, the frame is then separated into 3 separate images, (TL)top left, (TR)top right and (TM) top middle. For a visual representation of what is meant with this can be seen in Figure 5. These 3 images are resized to the dimensions of frame to a multiple of the models stride which in this case is to 448x448, because as mentioned before the YOLO-v5 framework only works with square images and the model is trained on this format which the same sort of resizing as training. This is also the step where downsampling can be applied the model, in subsection III-F the experiment for this can be found.

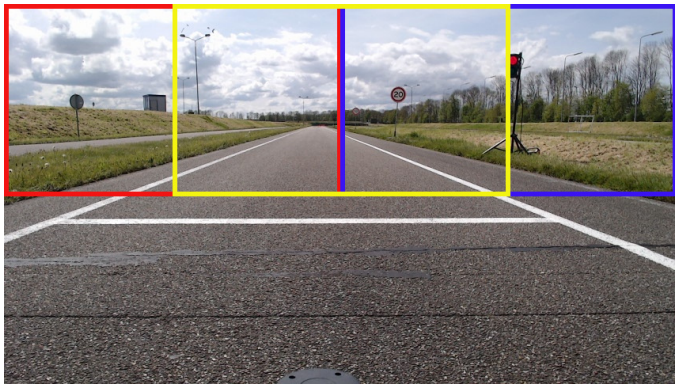

Fig. 5: This picture shows the images that are taken from a detection frame in the preprocessing stage of integration

3) Object detection Inference: The object detection is done with the help of the pytorch library and the YOLO-v5 framework, [Ult24b] by simple giving the loaded model an image with proper dimensions it will run inference and give an array of predictions. Then through processing these predictions with NMS(non maximum suppression), where based on the set based on Intersection over Union (IoU) and confidence thresholds will these turn these predictions into actual detections with according classes.

4) Post processing and distance estimation: After the detections are generated for all 3 of the sub-images, an offset is applied to all of their detections based on their original positions (top left, right, middle), to properly put them in the correct place. Then the distances to all of the objects are calculated based on the focal length described in subsection III-B. Detections that overlap between 2 sub images are filtered based on to which center of the sub image the detection is closer, which allows for a smooth transition between the sub images, allowing for example the tracking of a pedestrian throughout the whole image, while stile feeding the same input resolution before downsampling as it was trained on.

5) State Machine: The state machine is what will be read to make decisions from. The state is structured as a dictionary with the following elements:

- spotted red light: A boolean indicating if a red traffic light has been detected.
- Speed limit: An integer that represents the last detected speed limit. This updates every time it sees a new speed limit sign.
- Initial Person Position: A string that indicates the starting position of a detected pedestrian relative to the car. It can be ”None”, ”Left”, ”Middle” or ”Right”
- Starting Person Position: A string that indicates the current position of the pedestrian, it can have the same states as the initial person position.
- Car Spotted: A boolean indicating if a car has been detected within the distance threshold, which is used by a separate sub system to start an intake manoeuvre.

With these states, the system has enough information to simply alter its speed or stop.

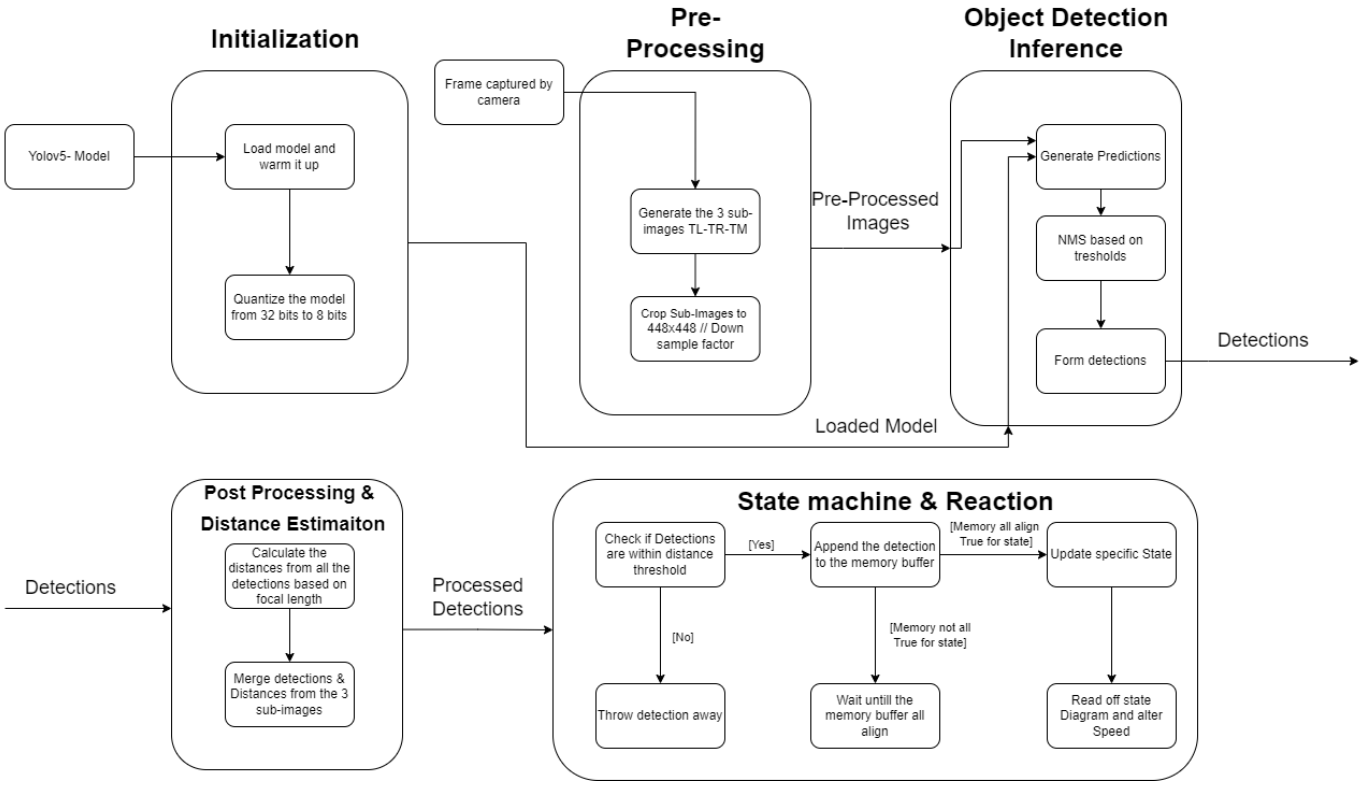

Fig. 4: In this diagram a whole flow chart of the steps of Integration can be seen including the sub-steps of the object detection system

For the state diagram to update on the right distances from objects, each individual class has a distance threshold.

- Red Light: 5m
- Pedestrian: 10m
- Speed Sign: 10m
- Car: 10m

To add another level of robustness, a memory buffer has been implemented, meaning that the system needs to see the same class within the distance threshold 5 times before it updates the state. The state is monitored with data loggers including the all the possible evaluation parameters.

After the state diagram is updated it has a shared variable with a different thread where it will give the most recent version of the state. Then based on the state the adjust throttle thread, will alter its speed based on a priority of objects, it will respond in order to red lights, pedestrians, Spotting cars, Speed limits.

6) Virtual test environment for Object detection: Keeping the end-goal requirements in mind of creating a system on a self-driven car, the system needs to be tested in real-time on the self driving car test track. Because real-time testing was limited to 1.5 hours per week (on a 6 week period) and because not just object detection, but other subsystems needed to be debugged, running extensive experiments on the real car was not viable. Thus, a virtual testing environment was created; it acts as a video player running the video data of an earlier recorded run, keeping the same resolution and FPS. It draws boxes of the detections that the model makes and shows the current state diagram in text. The virtual testing environment is ran on a spare NUC and also used in this way to measure CPU performance in the section III-F.

F. Optimization

The optimization is done through techniques such as quantization of model parameters and downsampling the input images of the system. To test the optimization performance, the accuracy, Precision, mAP:0.5, mAP: 0.5-0.95, FPS, RAM usage and CPU usage will be tracked for a set of optimization combinations. The goal is to see how effective the optimization methods are and how far we can push optimization until the system stops meeting the requirements. All accuracy metrics are tested through the YOLO-v5 framework val.py, with some adaptations to allow downsampling and quantization [Ult24b]. The speed metrics and seeing if it still meets the requirements of properly reacting to the objects are tested using the virtual testing environment. The speed metrics are run using the image merging mentioned in the integration section of the methodology. It should be noted that the accuracy metrics are run with a confidence threshold of 0. 001 %, which means that predictions of which the model is not very certain are also kept in as part of the calculations for the accuracy metrics, which could cause more false positives than if you were to put a higher confidence %, but this does allow evaluation

to better represent the performance of the model between different optimizations.

IV. Results and Discussion

This section will go over the resulting products and experiment results for the object detection system, including the annotated dataset, trained model, integrated system and results from optimization.

A. Dataset creation

The created dataset consists of a total of 6167 different images with labels after image augmentation, created from 211 different annotated images. The occurrence of each class throughout the dataset is shown in figure 6. This shows that the dataset is balanced enough to not cause any major bias towards a single class.

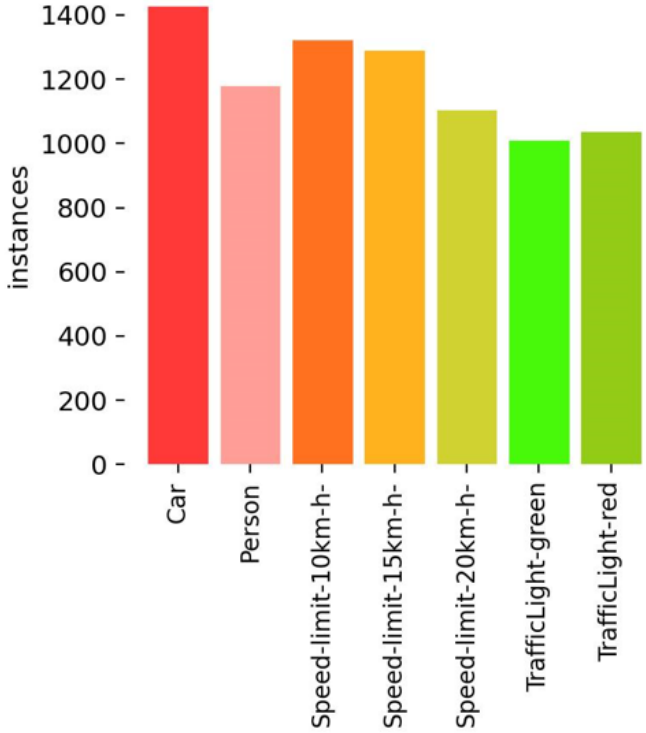

Fig. 6: This pictures shows the class count for each different class, showing a balanced distribution of classes

B. Model training

The model accuracy performance over the epochs can be seen in Figure 7. Both curves are logarithmic like, rapidly rising in the first few epochs but slowly flattening out in their growth. For all the training parameters please refer to the appendix at Figure 9.

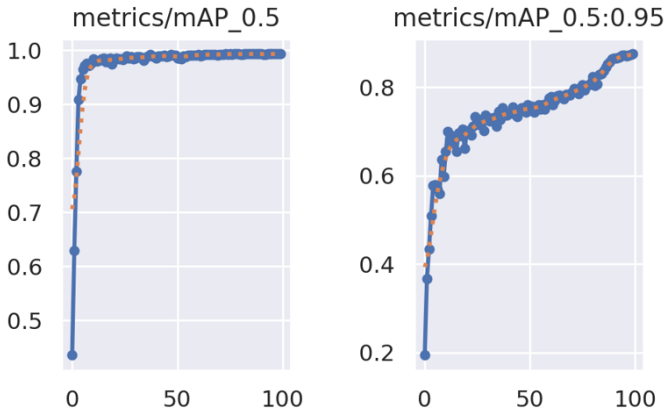

Fig. 7: This figure shows the MAP of 0.5(left) and 0.5:0.95(right) in a graph plotted against its epochs, showing a logarithmic like growth in MAP over the epochs

C. Integration

Integration is difficult to visualize through text or picture, so to see our live run during the RDW competition, refer to [Cha24]. Where it can be seen that during the competition that the car

- Stopped before a red light and went on green
- Switched its speed limit on the 20 and 10 sign
- Spotted the car and started to steer to the left, but did not finish the manoeuvre due to some other errors unrelated to object detection.
- Stopped too late for the second run second traffic light indicating that the distance estimation reaction thresholds were not tuned enough

This implies that all the requirements of accurately detecting classes, real-time processing, and integration were met. However, distance estimation was good enough to react only on objects that are actually close, but it was not perfect in the sense that the meters that you gave it as a threshold to stop for was not the same as the actual distance that it stopped. It was good enough for an estimate, but not for exact distances.

Besides not explicitly testing different for lighting conditions, the model was trained to handle it in the model training and dataset creation stages.

To see an example visualisation of what the system sees refer to Figure 8. This was created using the virtual testing environment.

TABLE III: Object Detection Performance and Efficiency Metrics

Model	Precision	Recall	$mAP_{0.5}$	$mAP_{0.5:0.95}$	FPS	CPU Usage (%)	RAM Usage (GB)	Meets Detection Requirements
Standard	0.985	0.99	0.993	0.874	11.92	55.65	0.62	True
Quantized (8 bits)	0.985	0.99	0.993	0.874	12.10	55.01	0.62	True
Quantized (8 bits) + Downsampled (factor: 1.2)	0.985	0.989	0.993	0.878	14.52	55.12	0.61	True
Quantized (8 bits) + Downsampled (factor: 1.5)	0.985	0.989	0.993	0.88	18.9	55.4	0.62	True
Quantized (8 bits) + Downsampled (factor: 2)	0.977	0.998	0.993	0.879	26.87	54.21	0.62	True
Quantized (8 bits) + Downsampled (factor: 2.5)	0.975	0.999	0.993	0.87	28.46	54.01	0.62	True
Quantized (8 bits) + Downsampled (factor: 3)	0.982	0.991	0.993	0.858	36.4	52.7	0.61	True
Quantized (8 bits) + Downsampled (factor: 4)	0.982	0.977	0.99	0.821	50.87	51.26	0.61	False
Quantized (8 bits) + Downsampled (factor: 6)	0.954	0.907	0.953	0.709	53.80	50.90	0.61	False
Quantized (8 bits) + Downsampled (factor: 8)	0.93	0.755	0.836	0.542	72.8	49.9	0.61	False

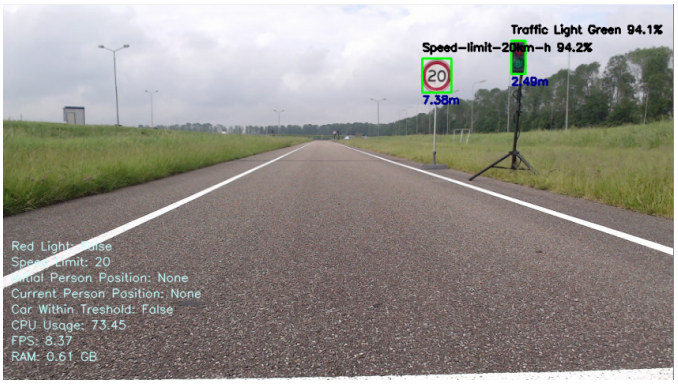

Fig. 8: A visualisation of what the model detects, the detections boxes are drawn as green rectangles around objects, where around it has the class that it detects and the distance calculated towards it

D. Efficiency

The data from the efficiency experimentation can be seen in table III. The accuracy parameters seem to stay close to constant until the downsampling factor of 3, where mAP 0.5-0.95 sees a drop in $\approx 2\%$, indicating that part of the validation set is no longer recognizable. The FPS of object detection is capped at 30 FPS, so as seen in table III CPU usage starts dropping after it passes the FPS cap as from there it can start to remain dormant between frames. Besides the parameters in the table, a metric called confidence % from 0-100% is given with each prediction, it differs between classes and occasion, but the more down-sampled the input image was the lower the average confidence rate, as with the standard model it was between 0.9 - 0.98, and when the down sample factor became as low as 4 it was already falling below the confidence threshold of 70 %, so to continue the experiment, the confidence threshold was dropped to 60% from down-sample factor 4 on to determine if detection requirements were still met. Which was met partially as it performed all the tasks correctly, expect seeing the speed sign of 10 as a speed sign of 15 once it can only see the corner of the speed sign, this confusion was originally filtered out through a high confidence threshold, as it was not certain with corners of speed sign. But now the corner of speed sign confidence % is as low as the normal detections, meaning that confidence % filtering with a global confidence % threshold is no longer possible. At a down-sample factor of 6 the pedestrian detector started functioning inconsistently, it could not fully keep track of the pedestrians positions during the whole crossing manoeuvre. At down-sample factor of 8 it fails all of the requirements of object detection as it fails to detect anything 5 times in a row to trigger the memory and change the state. The different jumps in FPS not being linear with the downsampling factor could be explained by that the image dimensions must always be a multiple of 32 (model stride), because after downsampling it is resized to the closest multiple of 32, which could slightly alter the actual downsampling factor being performed within the system.

V. Conclusion and Future Outlook

To conclude, the self driving car was capable to meet the accuracy, real time and integration requirements of the system fully, however distance estimation was only accurate enough that it can differ close from far objects, the estimated distances are not exact depending on the distance between 10-20% margin. Al though during testing days with the real self driving car there were different weather conditions and that this seemed to pose no issues, it was not tested directly with limited access to the testing faculty.

For optimization the decision was to choose of a combination of quantization and a downsampling factor of 2, to leave a margin of error during the competition. For future research on the self-driving car, these experiments should be ran on the real self driving car instead of in the virtual testing environment, as to see if it also keeps working in real life instead of just in the state diagram. Also an experiment can be ran testing the optimal FPS cap in combination with the optimizations, as this would allow a multi dimensional experiment which would be more time intensive to perform but allows to find to optimal parameters to lighten the load of the CPU in the NUC. Another optimization technique to consider in the future was to do the detections of all the merged 3 frames at the same time using batches instead of after each other, batches are used in CNN to detect multiple images at the same time as a single image, which can significantly speed up processing speed if done correctly. To finally conclude, this research and code can be expanded outside the self driving car testing track itself, these tools and frameworks are set up to be capable of creating a more broadly trained model capable of doing this tasks in the real world and perform different types of task besides the ones part of this track.

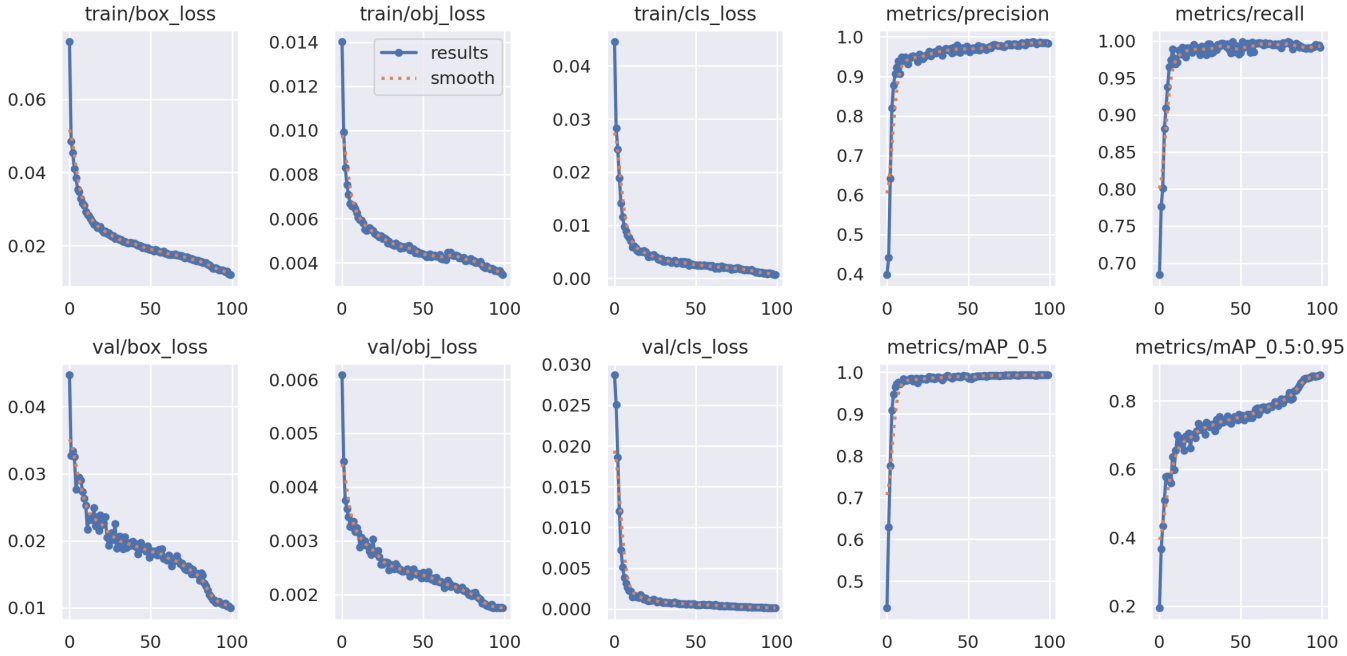

Fig. 9: This figure shows all of the different accuracy parameters for the training of the model over the epochs

VI. Appendix

A. GIT

Repository with all of the code used for the real time system, model training, dataset creation and virtual test environment, it also includes some other subsystems besides object detection under RealTime.

SDC Github

B. Training parameters

The full training parameters over time can be seen in Figure 9.

AI use statement

AI was used for a variety of things in this thesis. My most used AI was chatGPT-4o, its most important use was for brainstorming ideas, generating potential solutions as a starting point, a quicker way of scouring the internet and summarizing papers, worth mentioning is that before citing a paper used here, I carefully read the entire paper to ensure the summarized contents. I used it to create citation entries of different sources, but revised it and took full responsibility for it. I also used it to create initial drafts and examples of certain ideas in code, allowing me to quicker learn interactions with certain libraries and frameworks. But none of that code made it into the final product as code or text and I assume full responsibility for all the code and text in this thesis. I also used the auto vocabulary correcter of overleaf showing me if I misspelled any words.

References

[Mot39] General Motors. *Futurama Exhibit at the 1939 World's Fair*. Accessed: 2024-05-01. 1939. URL: https://example.com/futurama_exhibit.

[Jia+19] Licheng Jiao et al. "A Survey of Deep Learning-based Object Detection". In: *arXiv preprint arXiv:1907.09408* (2019).

[Ert+20] Christian Ertler et al. *The Mapillary Traffic Sign Dataset for Detection and Classification on a Global Scale*. arXiv:1909.04422v2 [cs.CV]. 2020. URL: https://arxiv.org/abs/1909.04422.

[Sin20] Amit Singh. *Solving Class Imbalance Problem in CNN*. Accessed: 2024-05-21. 2020. URL: https://medium.com/x8-the-ai-community/solving-class-imbalance-problem-in-cnn-9c7a5231c478.

[Nam22] Author Name. "An Edge Implementation of a Traffic Sign Detection System for Advanced Driver Assistance Systems". In: *Journal of Edge Computing* 10.2 (2022). URL: https://link.springer.com/article/10.1007/s41315-022-00232-4.

[Ami+23] Javeria Amin et al. "Pest Localization Using YOLOv5 and Classification Based on Quantum Convolutional Network". In: *Agriculture* 13.3 (2023), p. 662. DOI: 10.3390/agriculture13030662. URL: https://doi.org/10.3390/agriculture13030662.

[Cho+23] Ka Seng Chou et al. "A Lightweight Robust Distance Estimation Method for Navigation Aiding in Unsupervised Environment Using Monocular Camera". In: *Applied Sciences* 13.19 (2023),

p. 17. DOI: 10.3390/app131911038. URL: https://doi.org/10.3390/app131911038.

[RDW23] RDW. *Self Driving Challenge 2024 Information Document*. Accessed: 2024-05-01. 2023. URL: https://www.selfdrivingchallenge.nl/edition-2024.

[Cha24] Self Driving Challenge. *Self Driving Challenge 2024 - Final Race*. Accessed: 2024-06-24. 2024. URL: https://www.youtube.com/live/2dxOsTFqVdE?si=ZeszVv9vWupcZjiY&t=10646.

[Ins24] Ruhr University Bochum Institute of Neural Information Processing. *German Traffic Sign Recognition Benchmark (GTSRB) Dataset*. Accessed: 2024-05-01. 2024. URL: https://benchmark.ini.rub.de/gtsrb_dataset.html.

[Map24] Mapillary. *Mapillary Traffic Sign Dataset*. Accessed: 2024-05-01. 2024. URL: https://www.mapillary.com/dataset/trafficsign.

[Twe24] University of Twente. *Jupyter Wiki*. Accessed: 2024-05-01. 2024. URL: https://jupyter.wiki.utwente.nl.

[Ult24a] Ultralytics. *Train Custom Data*. Accessed: 2024-04-21. 2024. URL: https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/#local-logging.

[Ult24b] Ultralytics. *YOLOv5: Master README*. Accessed: 2024-05-01. 2024. URL: https://github.com/ultralytics/yolov5/blob/master/README.md.