Object Detection and Segmentation using Fisheye Images

Theodora Gaiceanu and Felix Slothower

LUND
UNIVERSITY

Department of Mathematics

MSc Thesis TFRT-9999
ISSN 0280–5316

Department of Mathematics
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Media-Tryck.
Lund 2023

Abstract

As computer vision becomes widely adopted, more applications demand a large effective field of view, something which is aptly accommodated by fisheye cameras. Autonomous driving vehicles, for example, require a full panorama view of their environment in order to navigate effectively. This requirement could be satisfied with as few as four fisheye cameras, one on each side of the car. However, due to the irregular compression of information at the edges of a fisheye distorted image, such a configuration is not well adapted for the convolutional neural networks traditionally used for computer vision tasks.

Rather than install more cameras with less severe distortion but who's field of view is significantly worse, we propose the use of tiled image rectification. Standard image rectification typically results in some of the field of view being cropped to maintain a consistent scale with respect to the center of the image. Tiling the rectification at different angles across the scene allows for the field of view to be preserved while still removing distortion. Using this method, we saw a significant improvement (mAP of 0.417 compared to mAP 0.318) over processing the distorted fisheye image while still observing the same field of view. We also tested traditional rectification and cylindrical rectification, both of which performed poorly with respect to the tiling and unprocessed methods.

Acknowledgements

This thesis project was carried out in the Spring of 2023 and hosted by Axis Communicatons AB. Originally intended as research concerning specific design challenges for one of Axis' clients, the project later evolved into a self-motivated study of the topic.

We would like to thank our supervisors at Axis, namely;

Angelo Delli Santi
Mikael Lindberg (currently at Saab)
Niclas Svensson
Anton Jakobsson

We would also like to thank our academic supervisors;

Senior Lecturer Mikael Nilsson
Assistant Professor Viktor Larsson
Professor Bo Bernhardsson

We appreciated all the help given by our supervisors, including their challenging suggestions. The time we spent during our meetings with them helped us improve our project, analyze our results better, and motivated us to give our best.

We would like to extend a special thanks to Maria Henningsson for her feedback and support, as well as to Linh Trang who supported us in the beginning of our thesis.

Theodora. I want to mention Felix for being an amazing thesis partner. I also appreciated very much the help, advice, and availability of the people from the university and from the company. I would like to thank my whole family for the love, support, and patience they had for me. Big thanks to my parents for always dreaming big towards my future and for encouraging me to achieve my goals. I want to also thank my husband-to-be for his love, encouragements, and for making me laugh when I needed it. Last but not least, I would like to express my gratitude to Professor Octavian Pastravanu from "Gheorghe Asachi" Technical University of Iasi, Romania for encouraging me to do a Master's Programme abroad.

Felix. First, a big thanks to my thesis partner Theodora, with whom it was always a pleasure to work with. I would also like to give thanks to our advisors, both at the university and at Axis, for thinking critically about our challenges and offering valuable feedback. I would also like to make a gesture of gratitude towards the departments of Mathematics, and of Controls at LTH. My interactions with the lecturers and professors were always very positive, and I appreciated the sense of community within their faculty.

Contents

1. Introduction **9**
- 1.1 Aim of the Thesis . . . 9
- 1.2 Fisheye Images . . . 9
- 1.3 WoodScape Dataset . . . 11
- 1.4 Analysis of the WoodScape Dataset . . . 12
- 1.5 Structure of the Thesis . . . 20

2. Background **21**
- 2.1 Object Detection with YOLO . . . 21
- 2.2 Object Detection for Fisheye Datasets . . . 26
- 2.3 Semantic Segmentation for Fisheye Datasets . . . 30
- 2.4 Amazon Web Services Elastic Compute Cloud . . . 31

3. Methods **32**
- 3.1 4th-Order Polynomial Mapping . . . 32
- 3.2 Cylindrical Mapping . . . 33
- 3.3 Rectilinear Mapping . . . 34
- 3.4 Tiled Mapping . . . 36
- 3.5 Intersection over Union (IoU) . . . 37
- 3.6 Mean Average Precision (mAP) . . . 38
- 3.7 F1 Score . . . 38

4. Object Detection **39**
- 4.1 Making WoodScape Compatible with YOLOv5 . . . 39
- 4.2 Conversion of the Bounding Box Coordinates . . . 41
- 4.3 Original Dataset . . . 43
- 4.4 Cylindrical Rectification . . . 49
- 4.5 Rectilinear Rectification . . . 54

5. Semantic Segmentation **59**
- 5.1 Pre-Processing the Annotations . . . 59
- 5.2 Original Dataset . . . 61
- 5.3 Cylindrical Rectification . . . 68
- 5.4 Rectilinear Rectification . . . 74

5.5 Analysis of Segmentation Results 79
5.6 Fair Model Comparison 83
6. Tiled Rectification **87**
6.1 Introduction 87
6.2 Adapting the Annotations 88
6.3 YOLO Results 89
6.4 Tile Merging 93
7. Conclusion and Discussion **98**
7.1 Summary of Experiments 98
7.2 Future Work 103
Bibliography **105**

1

Introduction

This chapter introduces the aim of the thesis and provides some details about fisheye images and the dataset used for this project. Lastly, the structure of the thesis is explained.

1.1 Aim of the Thesis

This thesis project aims to find improved methods of object detection and segmentation when using fisheye distorted images as input. The advantage of using wide-angle cameras over their conventional counterparts, is that the scene can be fully observed with a minimal number of sensors. However, in general, the performance of convolutional neural networks (CNN) on the distorted images is worse than the performances of the same model applied to standard images. This is because fisheye cameras introduce a significant nonlinear distortion, something which is not well suited for the translational invariance characteristics of conventional CNNs [1].

To combat this, most current implementations rectify the distorted image before being fed into an object detection network (e.g. rectilinear correction, piece-wise linear correction, cylindrical correction [1]). Image rectification has its issues, both decreasing the original field of view (FOV) and introducing resampling distortion [1]. We seek to explore methods which keep the original, wider FOV and that obtain better results than a classical model trained on fisheye images.

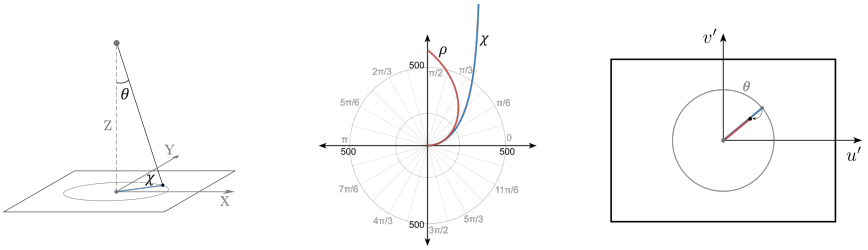
1.2 Fisheye Images

The first person to use the term *fisheye* was the physicist and inventor Robert W. Wood [2]. The term is related to the Snell's window phenomenon, which describes the 180° view of the world above water that a fish has when looking up towards the surface [2], [3]. An example of the phenomenon can be seen in Figure 1.1. Robert W. Wood also built a very simple fisheye camera which was based on a pin-hole camera filled with water.

Figure 1.1 The image is illustrative for the Snell's window phenomenon. The picture is taken from below the water surface, making the diver appear inside of the window [4].

As such, fisheye cameras are wide-angle cameras, some of them having a field of view (FOV) of around 180° or more [5]. This feature led the fisheye camera to be used in several computer vision applications, such as; generation of panoramic images [6], creating image-based virtual reality [7], self-localization problems [8], autonomous driving [9]. However, by the same token, the distortion introduced by the fisheye lenses introduces unique challenges when processing the 2D image [7], [1].

For the purposes of this project, the fisheye distortion was modeled using a fourth order polynomial. A high level illustration of the rectification process for a single image is provided in Figure 1.2, with supporting equations provided in Section 3.1. The key takeaway from these diagrams can be found in Figure 1.2b, where we see a non-linear relationship as angle θ increases between ρ and χ, plotted on polar coordinates. When θ is small, ρ and χ are nearly similar, representing little distortion at the center of fisheye image. But as theta increases, the deviation between ρ and χ represents the distortion seen at the far-periphery of a fisheye image. Examples of these images can be found in Figures 1.4 and 1.3.



(a) (b) (c)

Figure 1.2 Working from left to right, the rectification process, as parameterized in our project, has been illustrated. In the leftmost diagram, the 2D image is first projected to 3D and angle θ is measured for a given 3D pixel. In the middle diagram, a new radius ρ is calculated as a function of θ. Finally, the 3D pixel is projected back to 2D according to the ratio between χ and ρ as illustrated in the right diagram.

1.3 WoodScape Dataset

In our thesis, we use a public dataset for our experiments, called WoodScape [1]. According to the authors, WoodScape is the first fisheye dataset for autonomous driving. The dataset was extracted from four fisheye cameras offering a full 360° field of view (FOV). Figure 1.3 shows the orientation of the four cameras: front view (FV), rear view (RV), mirror view left (MVL), and mirror view right (MVR). All the images have the size 1280×966. Also, the dataset can be used for nine different applications, like object detection, segmentation, depth estimation, soiling detection. WoodScape has approximately 10,000 images and annotations for semantic segmentation and over 100,000 images and annotations for the other applications. The dataset contains annotations for both 2D and 3D bounding boxes, but we used only the 2D bounding boxes. Some samples from the datasets and possible applications are pictured in Figure 1.4.

A radial mapping function $r(\theta)$ was used to model the fisheye distortion, where r is the distance on the image from the centre of distortion, θ being the angle of the incident ray against the optical axis of the camera. The centre of distortion is defined as the place of intersection between the optical axis and the image plane, being at the same time the origin of the radial mapping function $r(\theta)$ [1].

The authors also provided a distortion correction method which uses a fourth order polynomial to model the fisheye distortion. The polynomial coefficients are provided for each camera angle to account for slight inconsistencies in the optics of each lens. The fourth order polynomial and supplementary equations can be found in Section 3.1.

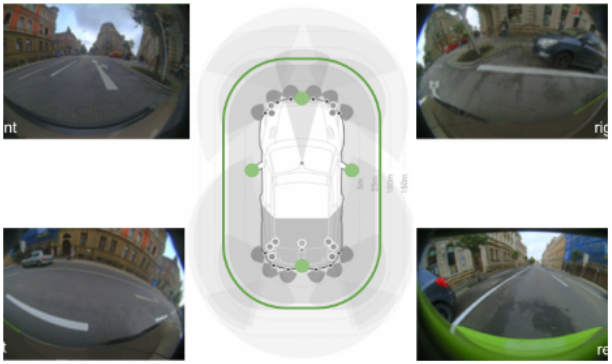


Figure 1.3 Placement of the four fisheye cameras that collected images from WoodScape dataset [1]

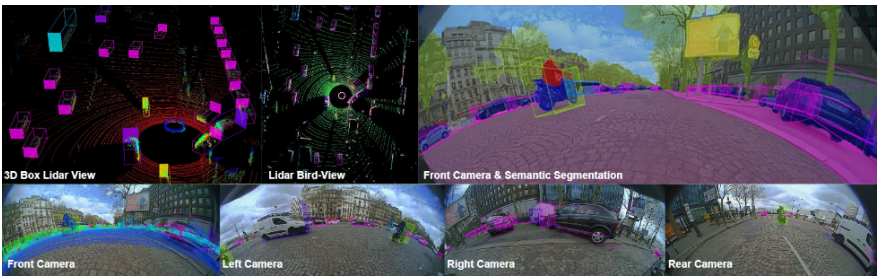


Figure 1.4 Image samples and possible application of WoodScape [1]

1.4 Analysis of the WoodScape Dataset

1.4.1 Objects in the Ground Truth

In order to better understand the performances of a model, it is important to evaluate how complex the problem is. Knowing the number of objects from the ground truth can help in this sense.

As the processed annotation files have as many lines as number of objects for both detection and segmentation tasks, an easy way to get the total number of objects in one image is to count the lines in its corresponding annotation file. For each task, one can make a dictionary having the name of the images as keys and the number of objects as value. The dictionary can be further processed in order to get the maximum number of objects in one image and the average number of objects for each task.

- Object detection with the original fisheye images

  The **average** number of objects per image for object detection with the original fisheye images is **8.75**, while the **maximum** number of objects is **35**.

The corresponding picture for the maximum number of objects can be seen in Figure 1.7.

- Object detection with the cylindrical corrected images

  The **average** number of objects per image for object detection with cylindrical corrected images is **8.75**, while the **maximum** number of objects is **35**. The corresponding picture for the maximum number of objects can be seen in Figure 1.8.

- Object detection with the rectilinear corrected images

  The **average** number of objects per image for object detection with rectilinear corrected images is **4.88**, while the **maximum** number of objects is **26**. The corresponding picture for the maximum number of objects can be seen in Figure 1.9.

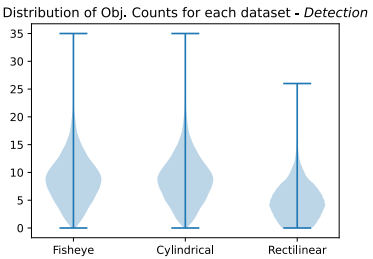


Figure 1.5 Violin plot showing the distribution of object counts in every image for the three object detection datasets used.

- Semantic segmentation with the original fisheye images

  The **average** number of objects per image for semantic segmentation with the original fisheye images is **46.42**, while the **maximum** number of objects is **173**. The corresponding picture for the maximum number of objects can be seen in Figure 1.10.

- Semantic segmentation with the cylindrical corrected images

  The **average** number of objects per image for semantic segmentation with the cylindrical corrected images is **44.78**, while the **maximum** number of objects is **169**. The corresponding picture for the maximum number of objects can be seen in Figure 1.11.

- Semantic segmentation with the rectilinear corrected images

  The **average** number of objects per image for semantic segmentation with the rectilinear corrected images is **30.77**, while the **maximum** number of objects

is **133**. The corresponding picture for the maximum number of objects can be seen in Figure 1.12.

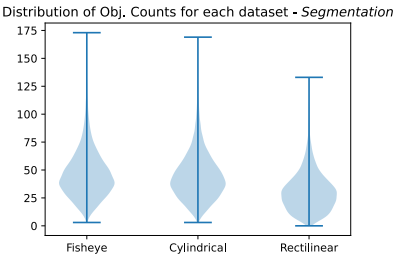


Figure 1.6 Violin plot showing the distribution of object counts in every image for the three segmentation datasets used.

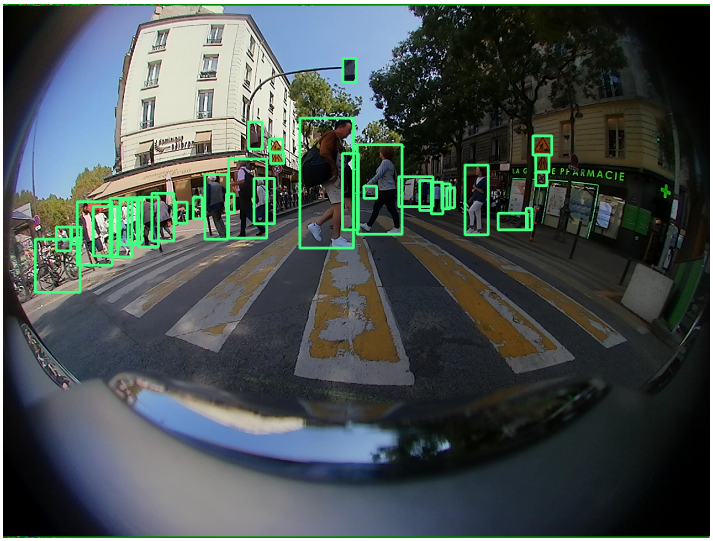


Figure 1.7 The fisheye image with the maximum number of objects for detection

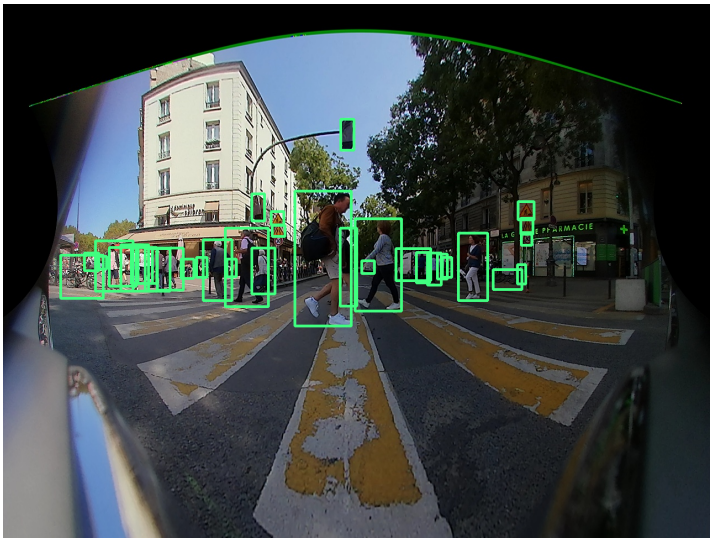


Figure 1.8 The cylindrical corrected image with the maximum number of objects for detection

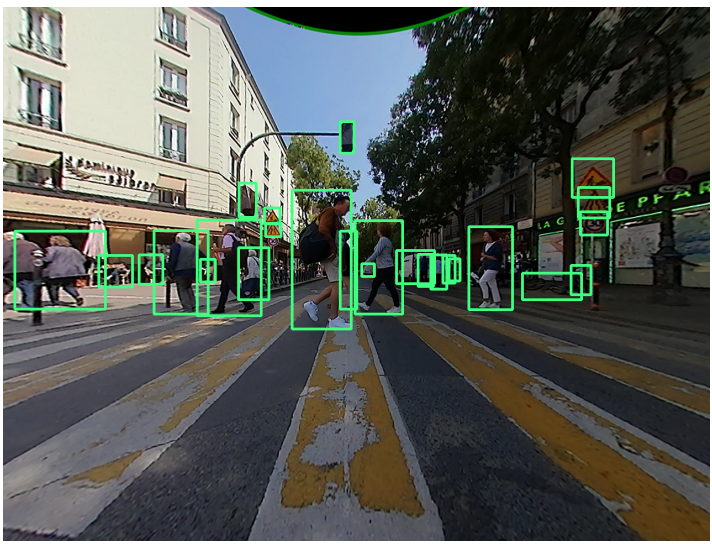


Figure 1.9 The rectilinear corrected image with the maximum number of objects for detection

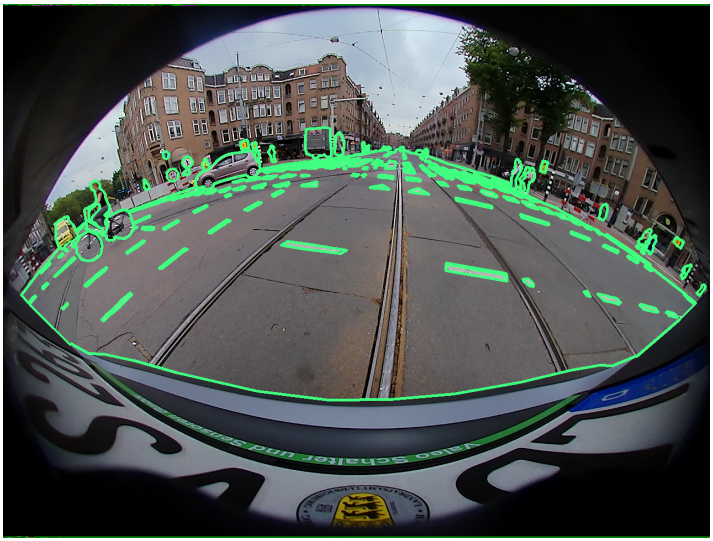


Figure 1.10 The fisheye image with the maximum number of objects for segmentation

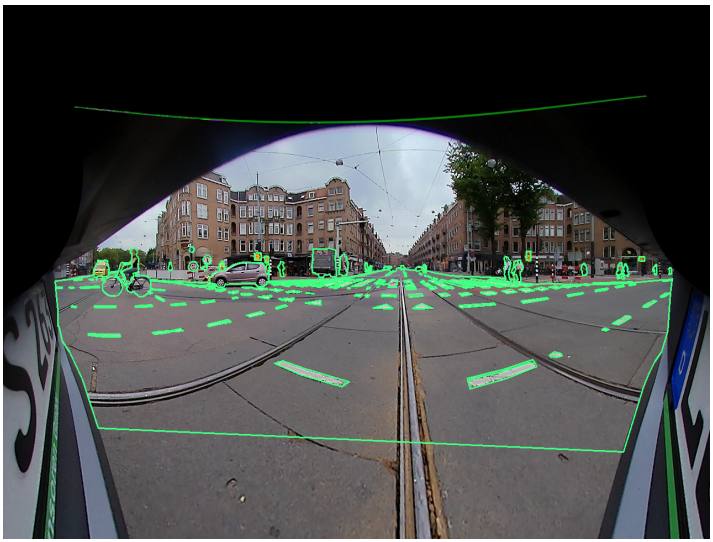


Figure 1.11 The cylindrical corrected image with the maximum number of objects for segmentation

1.4.2 Labelling Problems

When analyzing the dataset, we noticed that there are some missing annotations. It is unclear why this happened, but, for instance, there are people in the pictures

Figure 1.12 The rectilinear corrected image with the maximum number of objects for segmentation

that are not annotated. This has a negative impact on the detection and segmentation results, making the model more prone to classify objects as background.

Some examples of bad annotations can be seen in Figure 1.13 (inside the red rectangles). All the images have people that are not annotated accordingly, and Figures 1.13b and 1.13c have also some missing traffic signs. From a brief manual analysis of 20 randomly selected images, it was found that on average, 3.40 objects were missing a ground truth annotation per image. With respect to the average number of ground truth objects in each image, as discussed in the previous section, there is roughly a miss-annotation rate of 39%.

There are also certain edge cases where the existence of an object is not clear, either due to low resolution or occlusion. However, in spite of this low confidence, labeling may still be preferred as the model is ideally sensitive to cases where partially occluded humans are detected before walking in front of the ego vehicle like in Figure 1.14. This partially occluded person should be annotated, even if the annotator is not entirely sure whether it is truly a person or not. These are the edge cases that, when properly annotated, will make for safer autonomous driving. In the case of the referenced image, the person marked in red was not included in the ground truth.

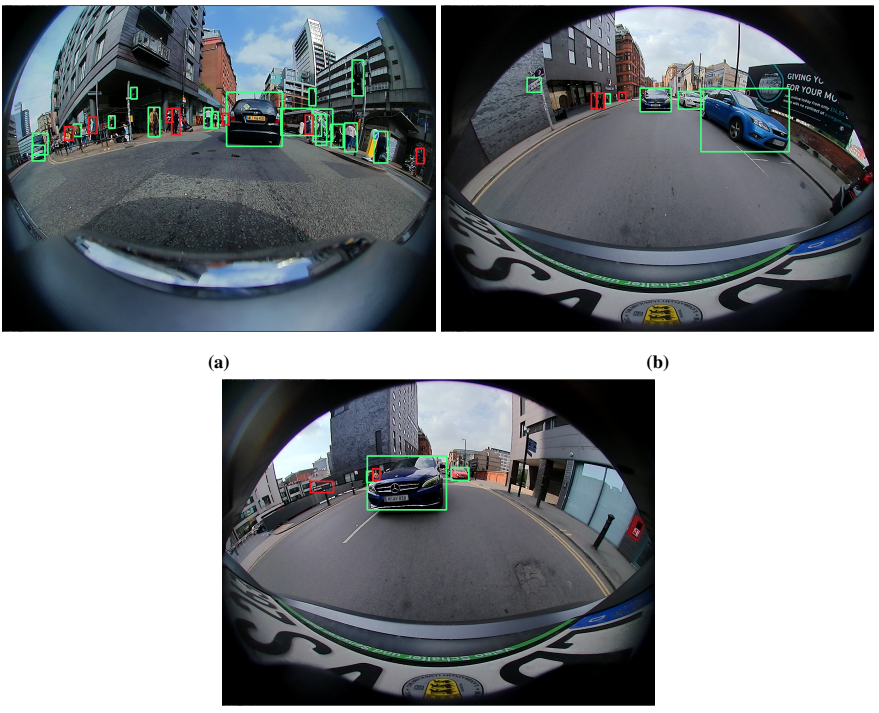


Figure 1.13 Examples of poor annotations from WoodScape. Bounding boxes in green are provided in the original dataset, while bounding boxes in red are missed instances of a particular class. Missed annotations in the ground truth can send mixed signals to the model during training and result in an overall worse performance.

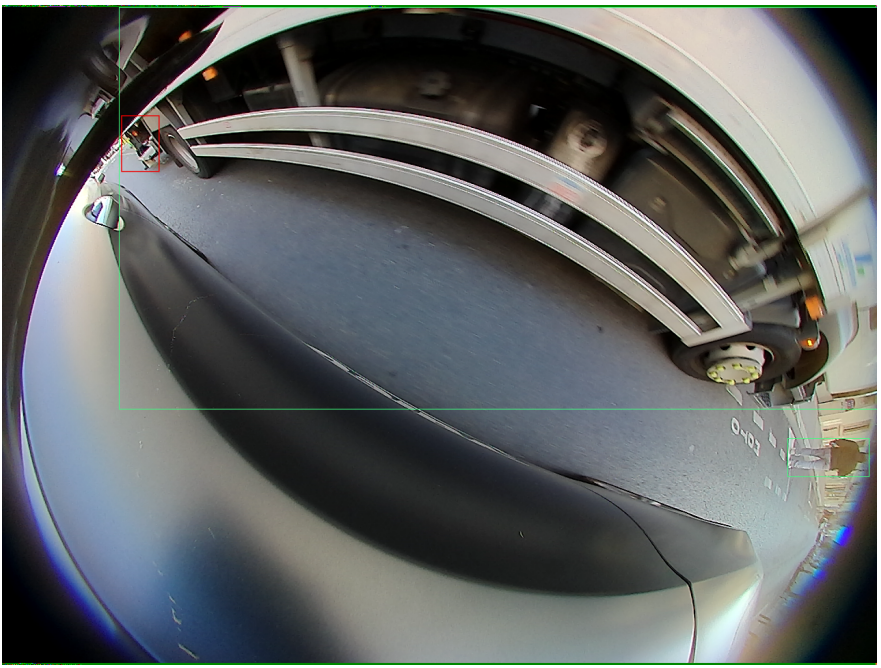


Figure 1.14 Notice the partial occlusion of a person near the top left of the image, standing behind the large truck. Because the object is small and occluded, we cannot be completely sure of what it is. However, from experience many drivers will naturally assume this could be a person, since often people are found moving things out the back of an open cargo truck.

1.5 Structure of the Thesis

The thesis is structured as follows:

Chapter 2: Explains the model that we used for training, introduces the state of the art approaches for detection and segmentation with fisheye images, and presents the software resources.

Chapter 3: Introduces theoretical notions used when implementing the rectifications and presents the considered metrics for evaluating the performance of our models.

Chapter 4: Shows the object detection results for the models using the fisheye, cylindrical and rectilinear corrected images.

Chapter 5: Shows the semantic segmentation results for the models using the fisheye, cylindrical and rectilinear corrected images. The chapter also includes an analysis of the segmentation results.

Chapter 6: Presents the segmentation results for the model using the tiled rectification.

Chapter 7: Contains the discussion and conclusions of our project, as well as some ideas for future work.

2

Background

This chapter explains what object detection and segmentation is, presents the You Only Look Once (YOLO) algorithm [10] that is used in our thesis for performing the object detection and segmentation tasks, and lists state of the art approaches for these tasks when having at input fisheye images. Finally, there is a short description of the virtual machine used to carry out the experiments.

2.1 Object Detection with YOLO

Object detection describes the task of localizing and classifying instances of different classes in digital images [11], [12]. The output is typically composed of the predicted class and a pair of coordinates for the bounding box that contains the instance [11], [12].

You Only Look Once (YOLO) is an object detection algorithm that gained a lot of popularity for being both fast and robust [13], [14]. For these reasons, we chose to use YOLO in our thesis. The detection problem is seen as a single regression problem, unlike other systems, like R-CNN [15], that have one part predicting the bounding boxes and another performing the classification task on these boxes. In short, YOLO first rescales the input image, then uses a convolutional neural network to predict the bounding boxes and the class probabilities, and applies non-maximum suppression in the end [10].

YOLO has the ability to embed contextual information, unlike R-CNN, which has limited information about the context [15]. Also, YOLO can predict at the same time the bounding boxes for all classes in an image [10].

2.1.1 First Version of YOLO

In the initial version of YOLO [10], the input image is down-sampled to 448×448 pixels. The algorithm applies an $N \times N$ grid on the input image where N is the number of gridlines. The grid cell that contains the center of an object is the one in charge of detecting that object. Also, every grid cell predicts B bounding boxes and their corresponding confidence. The confidence is calculated as the IoU between

the predicted box and ground truth, and is zero if no objects exist in the cell. The predicted output has the following form: $[x, y, w, h, conf]$, where (x, y) are the center coordinates of the bounding box w.r.t. the associated grid cell boundaries, w, h are the width and height represented as a fraction of the original image dimensions and finally $conf$ is the confidence score. In addition to these five output values, every grid cell also gives C conditional class probabilities, $P(Class_i|Object)$. A single set of class probabilities is predicted for each grid cell, no matter what is the number of bounding boxes B. One final step is to compute the class-specific confidence score for every box. This is done by multiplying the conditional class probabilities and the individual box confidence scores, as it can be seen in Equation 2.1

$$P(Class_i|Object) * P(Object) * IOU_{pred}^{truth} = P(Class_i) * IOU_{pred}^{truth}. \quad (2.1)$$

Figure 2.1 summarizes the detection process.

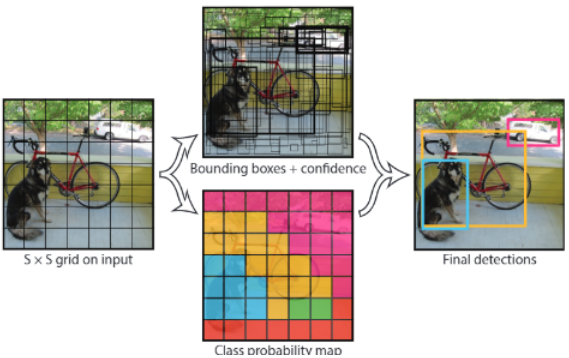


Figure 2.1 Detection process in YOLO [10]

The model consists of 24 convolutional layers extracting the features from the input image and 2 fully connected layers predicting the bounding box coordinates and output probabilities [10]. The model is based on GoogLeNet [16], an image classification model. A key difference, however, is that the inception modules of GoogLeNet were replaced by 1×1 reduction layers and 3×3 convolutional layers. Figure 2.2 presents the architecture of YOLO.

The authors pretrained the convolutional layers on the ImageNet classification dataset [17] using 224×224 pixel input images. The model is then modified for detection, the input resolution is increased to 448×448 pixels and final training is carried out.

2.1.2 Second Version of YOLO

The authors then developed YOLOv2 [18], an improved version of YOLO. They added batch normalization for every convolutional layer in YOLO. In addition, the

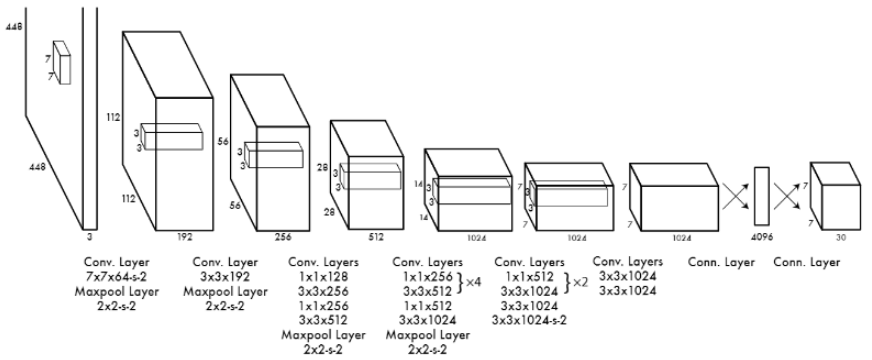


Figure 2.2 A schematic overview of the architecture of YOLO [10]

model was pretrained with higher resolution classifiers, using 448×448 pixel input images. Moreover, the fully connected layers in YOLO were replaced with anchor boxes. This change implied that the model predicted the class and objectness (confidence) for each anchor box.

The anchor boxes are basically priors for the bounding boxes. In order to find suitable priors, the authors performed k-means clustering on the dimensions of the bounding boxes from the training data. From multiple experiments, the authors found that a suitable value for k was 5 [18]. In other words, k-means is used to cluster the ground truth bounding boxes from the training set and determine the optimal set of anchor boxes to be used.

The location coordinates for the bounding boxes are generated relative to the grid cell center. Assuming one has the (c_x, c_y) cell offset from the upper left corner of the image, the anchor boxes p_w, p_h, and the 5 coordinates t_x, t_y, t_w, t_h, t_o predicted by the network, then the predicted bounding box coordinates, dimensions, and confidence score are computed as

$$b_x = \sigma(t_x) + c_x, \tag{2.2}$$

$$b_y = \sigma(t_y) + c_y, \tag{2.3}$$

$$b_w = p_w e^{t_w}, \tag{2.4}$$

$$b_h = p_h e^{t_h}, \tag{2.5}$$

and

$$P(Object) * IOU(b, Object) = \sigma(t_o), \tag{2.6}$$

where σ is the sigmoid function. The computations are also illustrated in Figure 2.3.

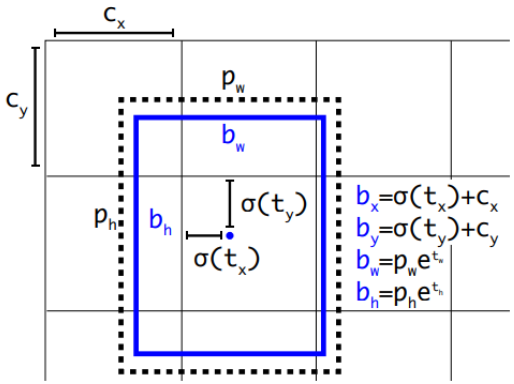


Figure 2.3 Computation of the bounding boxes using anchor boxes, as is done in YOLOv2 [18]

Another improvement from the first version is that YOLOv2 is trained on different image resolutions, making it more scale robust. To enable this, the image resolution during training changes every 10 batches. The size varies between 320×320 and 608×608 pixels at 32 pixel increments. This has the advantage that the same network can be used for performing detection at different image scales [18].

Moreover, YOLOv2 is based on Darknet-19 classification model [18]. Darknet-19 contains 19 convolutional layers and 5 maxpooling layers. The model uses mainly 3×3 filters, doubles the number of channels after each pooling step, uses global average pooling for making predictions, and performs batch normalization.

2.1.3 Third Version of YOLO

In YOLOv3 [19], 4 coordinates are predicted for every bounding box, instead of 5 (like in YOLOv2). These 4 values are t_x, t_y, t_w, t_h. Given these 4 coordinates, the (c_x, c_y) cell offset from the upper left corner of the image, the bounding box prior with dimensions p_w, p_h, the predictions are b_x, b_y, b_w, b_h and they are also computed as in YOLOv2. In addition, an objectness score for every bounding box is also predicted using logistic regression. The score is 1 when the bounding box prior overlays a ground truth object by more than the other bounding box priors. There is one bounding box prior per every ground truth object.

Similar to YOLOv2, YOLOv3 can be used to make predictions for 3 distinct scales, dividing the input image into a 13×13, 26×26 or 52×52 grid. Each of the 3 possible scales are used in different parts of the YOLOv3 network. The model is therefore able to make $N \times N \times [B \times (4 + 1 + no_{classes})]$ predictions, where N is

the grid scale, B is the number of boxes, and $no_{classes}$ is the number of considered classes [19].

In order to extract features, Darknet-53 network is used [19]. It contains 53 convolutional layers, and it is a hybrid between the Darknet-19 network from YOLOv2 and a residual network. Figure 2.4 illustrates the layers used in Darknet-53. Even though Darknet-53 is more complex than Darknet-19, it is faster than ResNet-101 and ResNet-152 (described in paper *Deep Residual Learning for Image Recognition* [20]).

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 2.4 The structure of Darknet-53 network [19]

2.1.4 Latest Versions of YOLO

There are ongoing improvements being made to the YOLO algorithm, with new versions of the model becoming available. Our thesis uses YOLOv5 from Ultralytics [21], which is similar to YOLOv3, by being able to detect objects of varying sizes using different scales, but which also facilitates data enhancement and training with custom datasets [13], [21].

Moreover, Ultralytics provides 5 versions of YOLOv5: nano (n), small (s), medium (m), large (l) and an even larger version (x). The nano version is the smallest and the fastest, designed for mobile solutions and being 2.5 MB in INT8 format. The small version has around 7.2 million parameters, the medium version consists of 21.2 million parameters. The large version has 46.5 million parameters and is designed especially for datasets with small objects. The largest version is the slowest (but also the most accurate) and has 86.7 million parameters.

Another new feature of YOLOv5 is the use of genetic algorithms (instead of just k-means) to generate the anchor boxes. The process is called autoanchor and,

in short, the anchor boxes are recomputed in case the default ones do not fit the data. The k-means algorithm is used together with the genetic one to create k-means evolved anchor boxes. This helps YOLOv5 perform well on different datasets.

Last but not least, YOLOv5 uses mosaic augmentation during training, combing 4 different images into one, which makes the model learn difficult and diverse cases.

2.2 Object Detection for Fisheye Datasets

Most current object detection implementations rectify the distorted image before being fed into an object detection network (e.g rectilinear, piece-wise linear or cylindrical rectification [1]). Image rectification has its issues, both decreasing the original field of view and introducing resampling distortion [1]. There are, however, other techniques that try to adapt the algorithm to the distortion, without rectifying the images.

An approach to increase the performance of object detection algorithms with fisheye images as input is to create novel annotations, which adapt to the distortion. An example of a research paper that follows this approach is *Generalized Object Detection on Fisheye Cameras for Autonomous Driving: Dataset, Representations and Base-line* by H. Rashed, E. Mohamed, G. Sistu, et al [22]. The authors created a new curved bounding box model adapted to fisheye distortion and developed their own dataset containing 10,000 fisheye images with annotations for several object representations.

Different representations for the bounding boxes were tested including; standard rectangular bounding boxes, oriented boxes (similar to the standard representation, but with an additional rotation parameter $\widehat{\theta}$), ellipse representation.

The authors also tried some distortion aware representations. The dataset used is WoodScape [1], which was created using fisheye cameras calibrated using a 4^{th} order polynomial model for the distortion. Using this information, the authors developed a new curved bounding box representation based on circular arcs. This representation is automatically generated, taking an object contour as input.

Lastly, the authors analyzed some generic polygon representations for the boxes, like uniform angular sampling, uniform perimeter sampling, and curvature-adaptive perimeter sampling.

In order to test their representations for the bounding boxes, the authors needed to adapt YOLOv3. They named their adapted model FisheyeYOLO. First, they replaced the Darknet53 model from YOLOv3 with ResNet18 model [20]. FisheyeYOLO is also able to perform object detection at different scales. Non-Maximum Suppression (NMS) [23] is used to select the highest confidence predictions. In addition, the authors used categorical cross-entropy and binary entropy losses instead of L_2 loss for categorical and object classification. A summary of FisheyeYOLO model is illustrated in Figure 2.5.

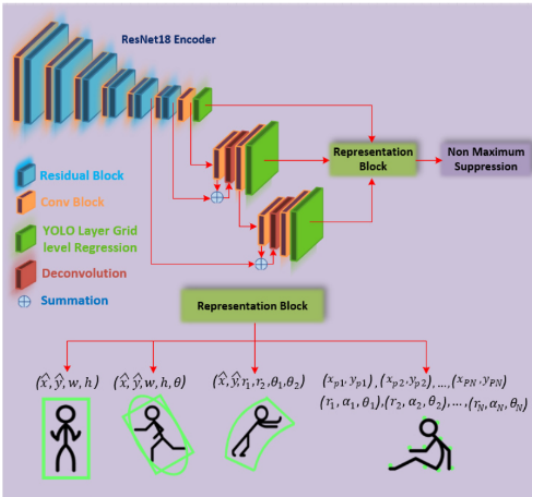


Figure 2.5 Overview of the architecture of FisheyeYOLO [22]

A different approach is to try to modify the CNNs in order to make them more robust to the fisheye distortion. An example in this regard is the Kernel Transformer Network (KTN)—a model able to adapt CNNs trained on standard undistorted images to 360^o image data [24]. At a high level, the KTN works by placing an additional convolution at every layer in a pre-trained CNN, which uses a unique kernel K_θ for every row. This effectively allows the KTN to learn the equirectangular distortion parameters while maintaing the original CNN's learned weights for object detection. Once trained, a KTN can be used for several source CNNs without any re-training.

The KTN can be used to substitute the standard convolutions in a CNN, as it is intended to be a generalization of it [24]. Normally, CNNs use the same kernel in order to extract the output feature map. In addition, the kernels should be translation invariant. However, this property is broken when using 360^o image data because of the distortion. But the kernels from KTN are aware of the distortion. Supposing we have an input feature map $I \in \mathbf{R}^{HxWxC}$ and a source kernel used in standard undistorted images $K \in \mathbf{R}^{kxkxC}$, the usual way to apply the source kernel is

$$F[x,y] = \sum_{i,j} K[i,j] * I[x-i, y-j]. \tag{2.7}$$

In contrast, KTN learns a function f that generates several kernels for several distortions

$$K_\Omega = f(K,\Omega), \tag{2.8}$$

$$F[x,y] = \sum_{i,j} K_{\Omega}[i,j] * I[x-i, y-j], \tag{2.9}$$

where Ω parameterizes the distortion. Since the distortion in 360^o images depends on the location, Ω can be expressed as

$$\Omega = g(\theta, \phi), \tag{2.10}$$

where θ and ϕ are the polar and azimuthal angle in spherical coordinates.

The authors of the paper state that one needs one KTN layer for every layer of a source CNN [24].

The topic of deformable kernels for fisheye images is also presented in *Fisheye-HDK: Hyperbolic Deformable Kernel Learning for Ultra-wide Field-of-view Image Recognition* by O. Ahmad and F. Lecue [25]. The authors introduce the hyperbolic deformable kernel, which is meant to be placed in the convolutional architecture of a traditional CNN. Its parameters are trainable and therefore does not require a prior knowledge of the camera and lens attributes. An assumption is made between the analogous representation of fisheye distortion and the hyperbolic space, or more specifically; a Poincaré ball [26]. With this, the model no longer needs to perform any projections.

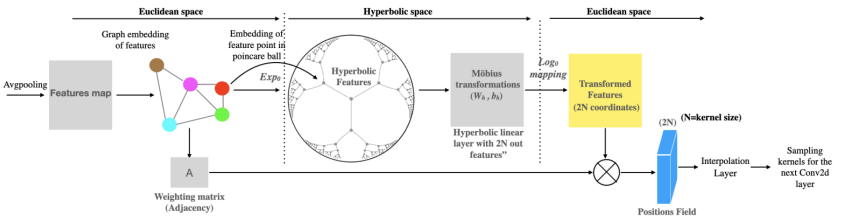


Figure 2.6 Architecture of the Fisheye HDK network [25]

From Figure 2.6 we see a high level schematic of the CNN architecture. The model first performs a graph embedding and then a hyperbolic embedding of feature points. The Möbius transformation [27] holds all the trainable parameters in the hyperbolic space. Then the points are extracted back to a Euclidean space via a $\log_0$ mappings and recombined with the former graph representation's weighting matrix **A**.

The result of all of this complexity however is rather impressive compared to other methods which have been tested. The model is able to produce a 16% improvement in mean accuracy (mAcc) score when compared to a 'vanilla' CNN (both trained on synthetic fish eye dataset with 10k images and tested on real world

car fish eye data of 800 images).

Paper *Restricted Deformable Convolution based Road Scene Semantic Segmentation using Surround View Cameras* [28] implements a deformable convolution similar to the previous paper ([25]). The key difference being that they do not include an embedding into the hyperbolic space, and they also fix their central point in the deformable convolution, giving them the name 'restricted deformable convolution'. The last λ layers of the network employ the deformable convolution operation, and it was found that with seven of these layers they were able to get the best results.

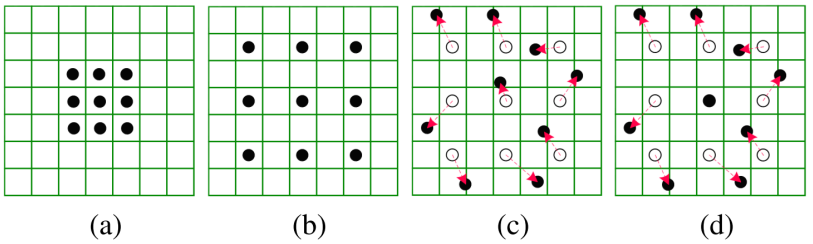


Figure 2.7 Different kernel representations: (a) represents a traditional 3x3 kernel. (b) is a 3x3 kernel with dilation 2. (c) is a deformable convolution with learnable offsets for each sampling point, illustrated as the red arrows. (d) is similar to (c), but only with a fixed central point (i.e. restricted deformable convolution). [28]

It is also worth mentioning that there is a difference between deformable convolution and a deformable kernel, as discovered when reading *Deformable Kernels: Adapting Effective Receptive Fields for Object Deformation* [29]. Deformable convolution specifies the convolution operation having a set of learned offsets for each sampling point. Deformable kernels on the other hand use rigid convolution sampling (no offsets), but the kernel itself can change for each sampled pixel.

Another technique is to make rotation-aware bounding boxes, especially in the case of overhead cameras. An example in this sense is *RAPiD: Rotation-aware People Detection in Overhead Fisheye Images* [30]. In [30] a rotation-aware people detection tool was developed, the boxes being arbitrarily-oriented depending on the people's positioning. In addition, a convolutional neural network was used to regress the angle of every bounding box by making use of a periodic loss function. Moreover, the authors built their own dataset with challenging scenarios that includes spatio-temporal annotations of rotated bounding boxes (the bounding boxes of the same person do not have a different ID in consecutive frames).

The paper [30] uses arbitrarily-oriented bounding boxes in order to detect people in fisheye distorted images. The used dataset has a top-down view.

The authors developed a new algorithm for detecting people in overhead, fisheye images. Their solution, named RAPiD, consists of a one-stage convolutional neural network that outputs the center coordinates, size, and angle of the arbitrarily-rotated bounding boxes of people. The model is based on YOLOv3. The predicted bounding box has the following format: $(\widehat{b} = (\widehat{b_x}, \widehat{b_y}, \widehat{b_w}, \widehat{b_h}, \widehat{b_\theta}, \widehat{b_{conf}})$, where $\widehat{b_x}, \widehat{b_y}$ are the predicted center coordinates, $\widehat{b_w}, \widehat{b_h}$ denote the predicted size (width, height), $\widehat{b_\theta}$ is the predicted angle of rotation, and $\widehat{b_{conf}}$ is the confidence score.

In order to be able to predict a rotation aware bounding box, the authors also developed their own loss function. The loss function is also based on the one used in YOLOv3, with the addition of the bounding-box rotation-angle loss. The angle loss was necessary because regression functions based on *L1* or *L2* distance are not aware of the periodicity of the angle, which can give wrong cost values due to the symmetry in the parametrization of rotated bounding boxes. The first step was to consider a π periodic function with the following expressions: $l_{angle}(\widehat{\theta}, \theta) = f(mod(\widehat{\theta} - \theta - \frac{\pi}{2}, \pi) - \frac{\pi}{2})$, where $\widehat{\theta}$ is the predicted angle, θ is the ground truth angle, f is a symmetric regression function. Next, the authors defined $[-\frac{\pi}{2}, \frac{\pi}{2})$ as the ground truth angle and they also constrain the ground truth $b_w < b_h$. Lastly, they make sure that the predicted angle range is inside $(-\pi, \pi)$.

One aspect that is worth mentioning is that one needs to have a fisheye dataset annotated with rotated bounding boxes in order to reproduce this approach.

2.3 Semantic Segmentation for Fisheye Datasets

Semantic segmentation is similar to object detection, but instead of outputting the coordinates for a bounding box surrounding the object, the models that perform segmentation return the coordinates of the polygon that surrounds an object. In this way, there is a segmentation map [31] for each object in the ground truth. In other words, semantic segmentation associates semantic classes for pixel observations [32]. Unlike object detection, where a bounding box has only two pairs of coordinates, the number of coordinates defining an object can be different for each instance.

An example of paper a that does semantic segmentation with fisheye images is *Real-time Semantic Segmentation for Fisheye Urban Driving Images based on ERFNet* [33]. The authors present a novel CNN architecture based on Efficient Residual Factorized Networks (ERFNet) [34] that give good results using only synthetic fisheye data.

They highlight the benefit of using fewer wide angle cameras to capture 360 video of a vehicle's perimeter. The benefits here being a reduced computational load and naturally reduced complexity w.r.t. camera calibration and time syncing [33].

An orthogonal fisheye model was used at varying focal lengths on a pre-existing, semantically segmented datasets to synthetically create a fisheye dataset. The model was an ERFNet CNN with 4 additional pyramid pooling modules inserted before the original upsampling layers. The pyramid pooling method works by pooling the feature map in the previous layer into 4 different pooling sizes. Then compressed via convolution to have 1 layer of depth, then upsampled and stacked onto the end of the original feature map before continuing to whatever final process was being used in the network's architecture. This bit has code name 'PSP'.

According to [33], the ERFNet PSP was much better than 'vanilla' ERFNet. It took slightly longer to run real-time, and performed slightly worse when the fisheye distortion was high (or the focal length in the projection model was low, in turn giving high distortion).

The paper shows that creating synthetic data by distorting existing datasets (e.g. CityScape [35]) is rather effective. The technique introduced does not however yield a good result compared to abilities of the same object detection algorithms on 'classical' undistorted images. That is, the model from the paper only yields a 61% IoU when the results for undistorted 'classical' images is much higher [33].

2.4 Amazon Web Services Elastic Compute Cloud

In order to run our experiments, we decided to use a virtual machine. Amazon Web Services (AWS) platform has a service, called Elastic Compute Cloud (EC2), that permits the users to rent virtual machines and develop their applications [36]. The virtual machine is called an instance.

We chose to launch an instance in US West (Oregon) region as this region has several available virtual computers with GPU. The user also needs to select the Amazon Machine Image (AMI) when working with EC2. The AMI is an image that specifies the operating system, applications and other information required to launch a virtual machine [37]. We selected *Deep Learning AMI GPU PyTorch 1.13.0 (Amazon Linux 2)* AMI. This AMI has a Linux kernel and it is designed to be used in Deep Learning applications. We went for the PyTorch version because YOLOv5 is implemented using PyTorch. After choosing the AMI, one needs to specify the virtual instance type. We chose a P3 instance, which is recommended for high performance computation, suitable for Deep Learning applications [36]. To be more exact, we selected a p3.2xlarge instance, which has one GPU with 16 GiB of memory and a 2.3 GHz (base) and 2.7 (turbo) Intel Xeon E5-2686 v4 processor.

3

Methods

The chapter presents the theoretical background behind the implementation. In addition, the evaluation metrics are also introduced; intersection over union (IoU) [38], mean average precision (mAP) [39], and F1-score [40].

3.1 4th-Order Polynomial Mapping

There are several models used to compensate for the lens distortion in a camera, each one befitting a different type of distortion or design requirement. For this project, the lens is modeled as having a 4th-order polynomial distortion.

The process of rectification involves first projecting the 2D image to a 3D real-world space, then projecting the 3D points back to a 2D image. In our case the first projection is either rectilinear or cylindrical and the second projection back to 2D uses the 4th-order polynomial mapping.

The fisheye distortion is orb-like with heavy distortion near the edge but minimal distortion towards the image center. For our application, this mapping is used only to project 3D points to a 2D image, the equations for which are provided below

$$\chi = \sqrt{X^2 + Y^2}, \tag{3.1}$$

$$\theta = arctan2(\chi, Z), \tag{3.2}$$

$$\rho = \rho(\theta) = k_1\theta + k_2\theta^2 + k_3\theta^3 + k_4\theta^4, \tag{3.3}$$

$$u' = \frac{\rho X}{\chi} \quad \text{if } \chi \neq 0 \text{ else } 0, \tag{3.4}$$

$$v' = \frac{\rho Y}{\chi} \quad \text{if } \chi \neq 0 \text{ else } 0, \tag{3.5}$$

$$u = u' + c_x + \frac{\mathrm{w}}{2} - 0.5, \tag{3.6}$$

$$v = v' a_R + c_y + \frac{\mathrm{h}}{2} - 0.5, \tag{3.7}$$

where X, Y, Z are the 3D points and u and v are the 2D image points. c_x and c_y denote the image center w.r.t. the top left corner of the image. w and h denote the image's width and height, respectively. f and a_R are the focal length and aspect ratio of the camera lens.

3.2 Cylindrical Mapping

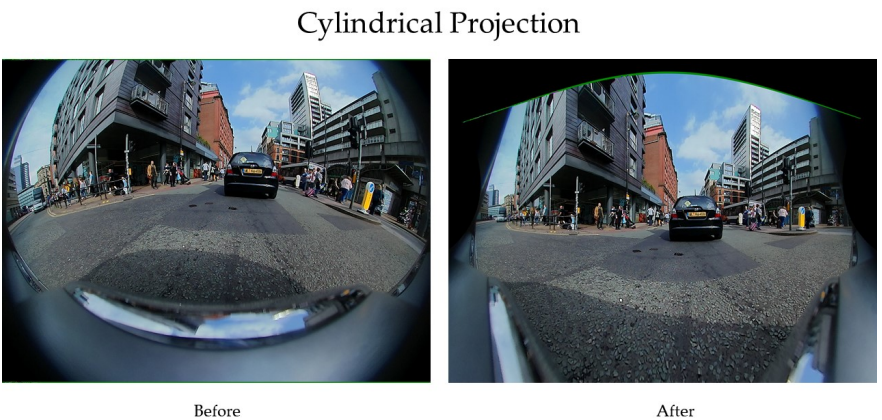


Figure 3.1 Before and after of a cylindrical projection. The vertical axis has clearly been undistorted, evidenced by the straightness in the height of all the buildings. The horizontal axis remains slightly distorted. This type of projection is helpful for maintaining the horizontal field of view while mitigating any vertical distortion.

The cylindrical mapping equations were used to project the 2D image points into 3D space. The result of combining this projection with the 4th-order polynomial mapping described above is an image who's vertical axis is now linear, but the horizontal axis maintains some level of radial distortion. The equations used to project the 2D image into 3D space are

$$\theta = \frac{u}{f}, \tag{3.8}$$

$$C = \frac{1}{\sqrt{v^2 + f^2}}, \tag{3.9}$$

$$X' = C * f sin\theta, \tag{3.10}$$

$$Y' = C * v, \tag{3.11}$$

$$Z' = C * f cos\theta. \tag{3.12}$$

Variable naming is the same as the previous section. The equations come actually from the WoodScape GitHub repository [41].

3.3 Rectilinear Mapping

A simple rectilinear mapping is used to project the 2D image into 3D space so that our 4th-order polynomial model can then correct the fisheye image distortion and yield an image which most closely resembles that from a pinhole camera. According to Hartley and Zisserman ([42]), the equation for rectilinear projection is

$$\begin{pmatrix} u \\ v \\ f \end{pmatrix} = \frac{f}{C} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}. \tag{3.13}$$

The value of the coefficient C was chosen such that the image projection had a reasonable scale and any black, seen at the top of the reconstructed image, was minimized. This projection distance worked well with the resulting 3D to 2D projection via the 4th-order polynomial described in Section 3.1.

In Figure 3.4 we see an example of a zoomed-out rectilinear mapping. This image highlights a critical issue in our projects goal of translation invariance and maintained wide field of view when processing barrel distorted or fisheye images. While this projection successfully demonstrates the mapping to a linear image space while maintaining a similar field of view, the scale and hence resolution of objects at the center of the image suffer greatly. When the rectilinear projection is applied, an object at a given distance Z from the camera's reference point (see Figure 3.2) will have the same scale across the entire field of view. Because image resolution is not increased during projection, the objects near the center which are now adjusted in scale, become heavily compressed.

A related issue are the inconsistencies in pixel density with respect to the 3D-world across the image sensor, i.e. objects appearing at the center of the image are rendered with a higher pixel density than objects near the periphery. After the rectilinear projection occurs, there are two options for handling resolution. Either the

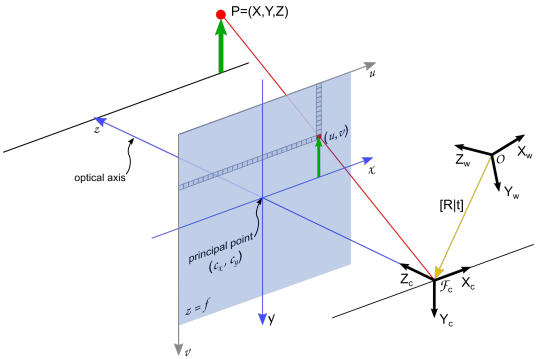


Figure 3.2 This figure illustrates the projection of a point on the image plane to 3D space. It has been taken from NVIDIA's Vision Programming Interface documentation page.

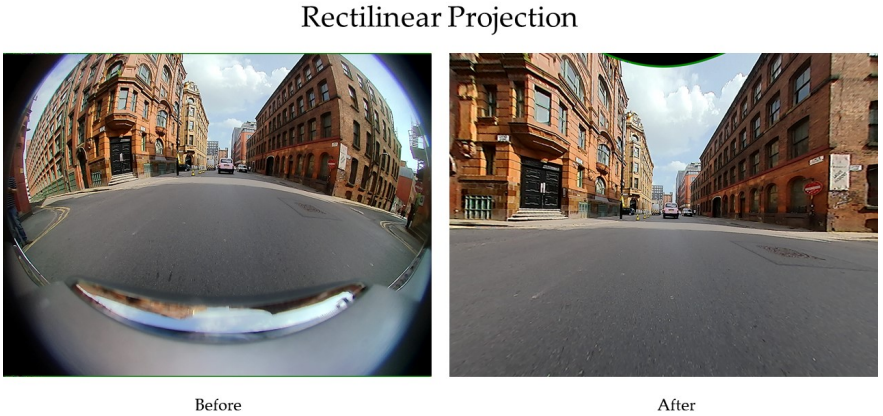


Figure 3.3 The before and after results from our rectilinear projection. Much of the image has been cropped from the original, dramatically reducing the field of view. The 4th-order polynomial mapping is successful as evidenced by the straight edges of the buildings in the rectified image.

total image resolution must be increased to maintain the information being compressed at the center and allow for interpolation of pixel values at the periphery, or resolution is maintained and the objects at the center are heavily compressed. As mentioned, the variability in pixel density between objects rendered at the center and those on the far periphery becomes readily apparent.

Figure 3.4 Zoomed-out version of the rectilinear projection. The image demonstrates the deviation from the true transformation in the 4th-order polynomial function as we get closer to the images edges.

3.4 Tiled Mapping

The idea is to divide the original fisheye images into 3 or 6 tiles (depending on the camera orientation) and to rectify each tile individually. The code for generating the tiles was inspired by *LaMAR: Benchmarking Localization and Mapping for Augmented Reality* [43]. Each tile has its corresponding rotation angles. See the diagram in Figure 3.5 for an illustration of the car's reference coordinate system. By using this coordinate system and the corresponding angles (called x, y, z), one can build the rotation matrix as

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cox(x) & -sin(x) \\ 0 & sin(x) & cos(x) \end{bmatrix}, \tag{3.14}$$

$$R_y = \begin{bmatrix} cos(y) & 0 & -sin(y) \\ 0 & 1 & 0 \\ sin(y) & 0 & cos(y) \end{bmatrix}, \tag{3.15}$$

$$R_z = \begin{bmatrix} cos(z) & -sin(z) & 0 \\ sin(z) & cos(z) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.16}$$

$$R = R_x \times R_y \times R_z. \tag{3.17}$$

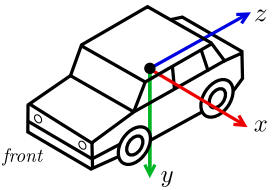


Figure 3.5 This diagram shows the car's reference coordinate system for the rear facing camera. The rotation matrix, R is defined with respect to this coordinate system. The z axis here is parallel to the forward driving direction and y is pointing straight down.

Camera	(x,y,z)
Mirrors	$x = (\alpha, \beta, 0)$
Front/Rear	$(\alpha, 0, 0)$

Table 3.1 A different set of projection angles (x,y,z) is used for the side-view mirror cameras versus the front and rear cameras.

This rotation matrix is used in order to undistort the tiles. Similar to the previous rectilinear correction, a mapping is used to project the 2D tile image into 3D space in order to correct the fisheye distortion and output a rectified image. The projection equations can be seen in Equations 3.18 and 3.19

$$C = 2f, \tag{3.18}$$

$$\begin{pmatrix} u \\ v \\ f \end{pmatrix} = \frac{f}{C} \begin{pmatrix} X \\ Y \\ \frac{1}{2}Z \end{pmatrix}. \tag{3.19}$$

3.5 Intersection over Union (IoU)

IoU (or Jaccard index) is a commonly used evaluation metric in object detection problems [38], [44]. Supposing that A is the area of the predicted bounding box and B is the area of the bounding box from the ground truth, then the IoU can be used in order to compute how much overlap there is between the two bounding boxes, relative to their combined size. As it can be seen in Equation 3.20, the IoU consists of the overlapping area between the two bounding boxes divided by the area obtained from the union of the two bounding boxes as

$$IoU = \frac{A \cap B}{A \cup B}. \tag{3.20}$$

3.6 Mean Average Precision (mAP)

The mean average precision (mAP) is another popular metric for evaluating the performance of object detection algorithms [45]. The mAP is the average of the AP for all predicted classes, where the AP is computed as the area under the precision-recall curve [39].

The precision is expressed as

$$Precision = \frac{TP}{TP+FP}, \tag{3.21}$$

and the recall is defined as

$$Recall = \frac{TP}{TP+FN}, \tag{3.22}$$

where TP is the number of true positives, FP is the number of false positives, and FN is the number of false negatives [46], [44].

First, every prediction is checked if it has an IoU higher than 0.5. If it does, the prediction is counted as a true positive, otherwise it is considered to be a false positive. Then, the precision and recall are computed for increasingly large subsets of predictions, from the most confident predictions to the least confident ones [39]. The precision-recall curve is plotted using the computed values. The AP is the area under the precision-recall curve. This is computed as a quadrature approximation sampling at 11 uniformly spaced values of recall [39]. This mAP is sometimes also called mAP50 (because of the 0.5 IoU threshold). One can also see mAP50-95, which corresponds to the average mAP for IoU threasholds from 0.5 to 0.95, normally with a step of 0.05.

3.7 F1 Score

Another metric that shows the accuracy of a test is the F1 score [40]. As it can be seen in Equation 3.23, the F1 score is expressed as

$$F1 = 2 \times \frac{P \times R}{P+R}, \tag{3.23}$$

where P is precision and R is recall. The higher the F1 score, the better. The range is between 0 and 1, a value of 1 meaning that both precision and recall are perfect.

4

Object Detection

This chapter presents the baseline and rectification experiments for the object detection task. Our thinking was to inspect the performances of classical object detection with fisheye images. Therefore, we thought that our baseline should take as input the distorted images from the WoodScape dataset [1] and their corresponding 2D bounding box annotations, feed them into the YOLOv5 model [21] and analyze the output.

The dataset is then rectified using two different mapping functions, cylindrical and rectilinear, and the same experiment is performed as with the original dataset. To do this, it was also necessary to map the ground truth annotations as covered in Section 4.2.

4.1 Making WoodScape Compatible with YOLOv5

We used 8234 RGB images and their corresponding 2D annotations for the baseline. The 2D annotations specify the x and y coordinates of the corners of the rectangle bounding boxes containing the objects. More precisely, with 2D annotations, the bounding box coordinates of the top-left (xmin, ymin) and bottom-right (xmax, ymax) corner are listed. There are five classes represented for 2D annotations: people, vehicles, bicycle, traffic lights and traffic sign. Each of these classes has a corresponding class ID: 0 - vehicles, 1 -person, 2 - bicycle, 3 - traffic light, 4 - traffic sign. In summary, the annotation format for the 2D boxes is the following: name of the class, class ID, xmin, ymin, xmax, ymax.

However, YOLO has a different annotation format. For each image, YOLO needs a text file that contains the annotations. The file contains as many lines as the number of bounding boxes in an image. Each line specifies the corresponding class ID of the object represented in the bounding box, the x and y coordinates of the center of the bounding box, as well as the width and the height of the bounding box. It is worth mentioning that the x and y coordinates, as well as the width and the height, are normalized.

Consequently, a script to do the conversion from the 2D WoodScape annotations to the YOLO annotations format was implemented. First, the script extracts the important information from each 2D WoodScape annotation file in a data dictionary containing: filename, image size, and a dictionary with the bounding boxes (for each bounding box, the object name, class ID, xmin, ymin, xmax, ymax are saved). Next, a function takes as input this data dictionary, and converts the information according to the YOLO format using the following expressions from Equations 4.1 - 4.4:

$$x = \frac{xmin + xmax}{2}, \tag{4.1}$$

$$y = \frac{ymin + ymax}{2}, \tag{4.2}$$

$$width = xmax - xmin, \tag{4.3}$$

$$height = ymax - ymin. \tag{4.4}$$

The parameters $x, y, width, height$ are further normalized. After doing all the computations, the class ID, $x, y, width$, and $height$ of every bounding box are saved in a text file, having the same name as its corresponding image.

For example, supposing we have the following 2D WoodScape annotation file:

```
vehicles,0,985,317,1047,369
person,1,191,346,243,428
traffic_sign,4,400,231,419,252
vehicles,0,656,255,695,286
vehicles,0,601,254,638,282
vehicles,0,325,275,488,354
```

Listing 4.1 2D annotation file from WoodScape

Its corresponding annotation file in YOLO format is:

```
0 0.794 0.355 0.048 0.054
1 0.170 0.401 0.041 0.085
4 0.320 0.250 0.015 0.022
0 0.528 0.280 0.030 0.032
0 0.484 0.277 0.029 0.029
0 0.318 0.326 0.127 0.082
```

Listing 4.2 Adapted 2D annotation file from WoodScape

After converting all the annotation files to YOLO format, one needs to divide the dataset into training, validation, and testing. We implemented another script to divide the dataset into 80% training, 10% validation, and 10% testing.

One last thing worth mentioning is that WoodScape has only 5 classes and they are not included in the set of classes which YOLO has been pre-trained on. Therefore, one needs to add a YAML file which includes the path to the WoodScape images and the classes with their corresponding class ID.

4.2 Conversion of the Bounding Box Coordinates

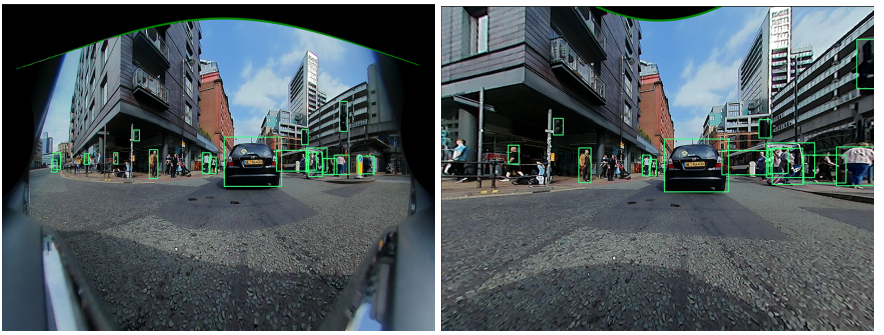
In order to analyze YOLO results with the corrected images, one needs to generate also the 2D bounding box annotations for the cylindrical and the rectilinear images. An example of an annotated fisheye image with 2D bounding boxes can be seen in Figure 4.1.

Figure 4.1 Example of an annotated fisheye image with 2D bounding boxes, taken directly from the WoodScape dataset.

The straightforward approach was to take the coordinates of the fisheye bounding boxes and make projections in order to get the corresponding coordinates for the cylindrical and rectilinear images. In detail, we used the two coordinates (top left and lower right corners) of each bounding box in the fisheye images and we performed the 2D to 3D projection in order to get the real-world space coordinates. Afterward, we used the real-world space coordinates and we performed a cylindrical/rectilinear 3D to 2D projection in order to get the two corresponding coordinates of the bounding boxes for the cylindrical/rectilinear corrected images. An example of the resulted coordinates can be observed in Figure 4.2.

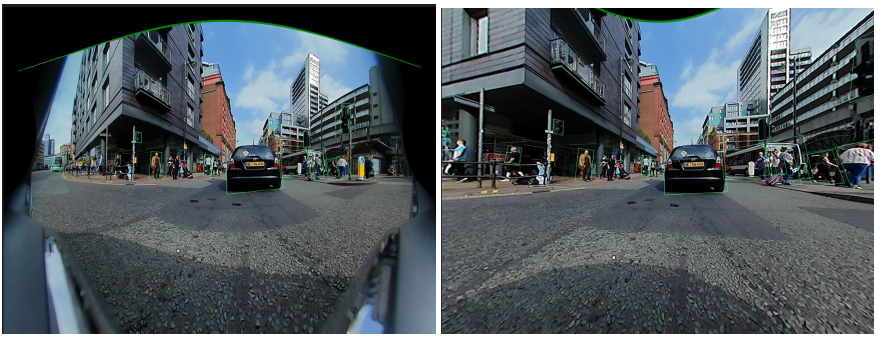
But by following this approach, we noticed that because of the distortion, the resulting bounding boxes did not cover the whole object, especially in the corners of the image. For example, in Figure 4.2a, the bounding boxes are cropping the bus and some people from the right part of the image. The same remark is applicable to Figure 4.2b.

Therefore, instead of assuming each bounding box is a rectangle, definable by only two coordinates, the bounding box was posed as a polygon defined by a co-



(a) (b)

Figure 4.2 Example of a cylindrical and a rectilinear image with the resulted bounding box coordinates after applying the projection considering two coordinates of the bounding boxes from the fisheye image. Notice the objects being cropped by the resultant bounding box, particularly in objects near the edge of the image in the rectilinear projection.

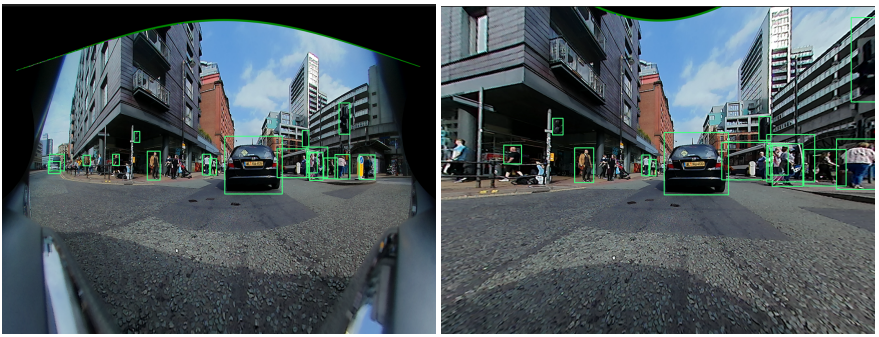


(a) (b)

Figure 4.3 Example of a cylindrical and a rectilinear image with the remapped bounding box coordinates after applying the projection considering four coordinates of the bounding boxes from the fisheye image. The resulting skew of the bounding box should be corrected in order to work better with the YOLO object detector (see figure 4.4).

ordinate at each corner. Remapping of coordinates followed the same procedure as before. An example of the remapped coordinates can be observed in Figure 4.3. As it can be noticed in Figure 4.3, the resulting four bounding box coordinates for the cylindrical and the rectilinear images do not form a rectangle, as they did for the fisheye image. This was expected because of the distortion correction.

However, YOLO can only take two coordinates for each bounding box, not four. Consequently, an additional processing step was performed: given the resulted four coordinates of the bounding boxes for the cylindrical/ rectilinear corrected images, one takes into account only the (x_{min}, y_{min}) and (x_{max}, y_{max}) coordinates. The final



(a) (b)

Figure 4.4 Example of a cylindrical and a rectilinear image with the final bounding box annotations

bounding boxes can be seen in Figure 4.4.

As it can be observed in Figure 4.4, the bounding boxes for the cylindrical and the rectilinear images cover a larger area than the object. This effect can be more noticeable in the corners of the images, where the fisheye distortion is stronger. For instance, the bounding boxes for the traffic lights in the top right corner of Figure 4.4a covers a larger area. The same can be said for the bounding boxes of the bus and the car in the left part of Figure 4.4b.

4.3 Original Dataset

We first trained YOLO on the original fisheye distorted images from the WoodScape dataset using the pretrained COCO weights. We trained the model for 200 epochs using a batch size of 32. We initially considered the small model from YOLO. As it can be seen in Listing 4.3, the training took 2.470 hours. Listing 4.3 displays the validation results, where 821 images were used.

The dataset is imbalanced. As it can be noticed in Figure 4.5, there are significantly more vehicles than other classes. When working with imbalanced datasets, some good metrics for evaluation are the precision and the recall.

```
200 epochs completed in 2.470 hours
Model summary : 157 layers, 7023610 parameters, 0 gradients, 15 .8 GFLOPs:

                 Class  Images  Instances       p       R   mAP50  mAP50-95
                   all     821       6953   0.645   0.435   0.494     0.262
              vehicles     821       4367   0.752    0.66   0.733     0.476
                person     821       1521    0.78   0.447   0.579     0.292
               bicycle     821        680   0.507   0.282   0.324     0.155
         traffic_light     821        137   0.634   0.391   0.434     0.183
          traffic_sign     821        248   0.553   0.395   0.398     0.206
```

Listing 4.3 Object detection validation results of the original fisheye dataset from WoodScape using YOLOv5 as the training model - small model of YOLOv5. The picture shows for each class the number of images used for validation, the number of instances, precision (P), recall (R), mAP50 and mAP50-95 scores.

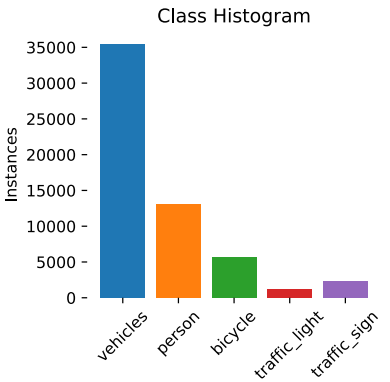


Figure 4.5 Distribution of class instances for object detection in the original fisheye images from WoodScape. The vehicles are the most represented, while the traffic lights are the least represented.

The results for precision and recall can be seen in Listing 4.3. Overall, the precision is 0.645 and the recall is lower, 0.435. The lower recall indicates a high false negative rate, which means that the model fails to predict some objects that are present in the image. The best results for recall (0.66) are for the vehicles, which was expected since this is the dominant class. The precision is the highest for the person class (0.78), and closely followed by the vehicles class (0.752).

Another metric that is frequently used when evaluating the results of object detection is mean average precision (mAP). The average precision (AP) makes use of a threshold value of the intersection over union (IoU). Some common thresholds values are 0.5, corresponding to an IoU over 0.5 and mAP50, and between 0.5 and 0.95, corresponding to an IoU between 0.5 and 0.95 and mAP50-95. Computing the AP corresponds to finding the area under the precision-recall curve. Figure 4.6 illustrates the precision-recall curve. The results on the right side of the image correspond to the mAP50 results. As it can be seen, the best result is for the vehicles,

having the largest area under the curve and a mAP50 of 0.733. The traffic sign and the bicycle classes have the worst results.

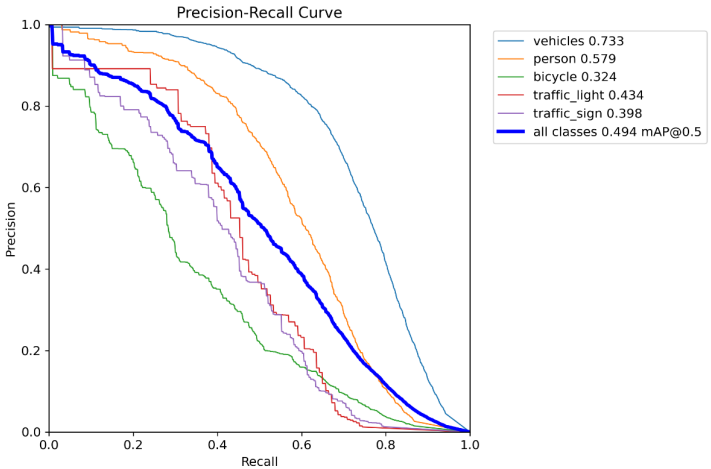


Figure 4.6 Precision-recall curve after performing object detection on the original fisheye dataset. The results on the top right corner correspond to the mAP50 scores. Vehicles (light blue) achieved the best results, while the bicycle class (green) performed the worst.

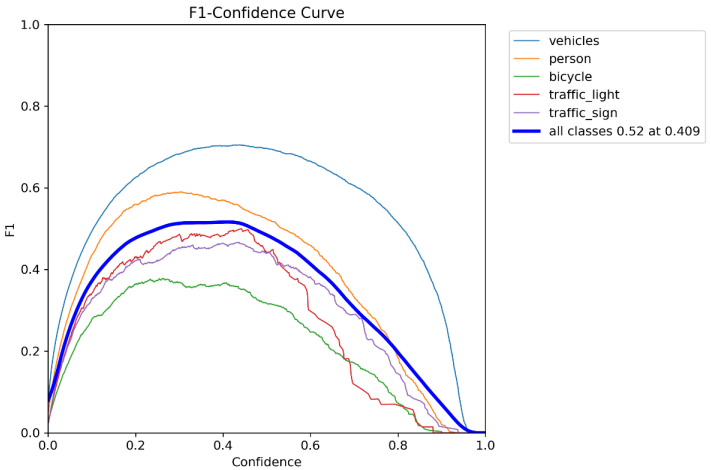


Figure 4.7 F1-confidence curve after performing object detection on the original fisheye dataset. The vehicles (light blue) have the highest F1 score, while the bicycle class (green) has the lowest F1 score.

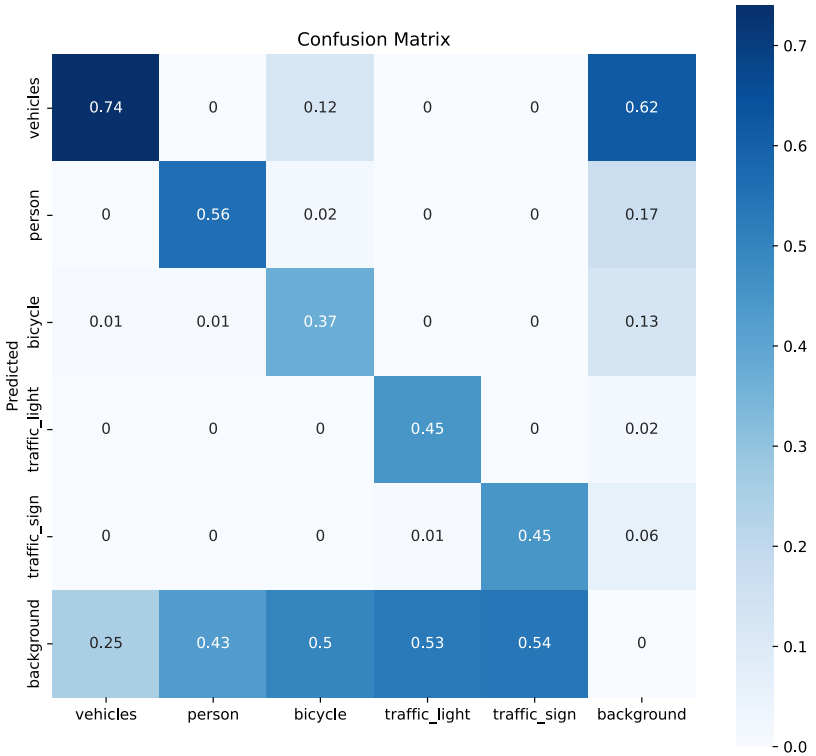


Figure 4.8 Confusion matrix after performing object detection on the original fisheye dataset. The vehicles have the highest true positive rate, while the bicycle have the lowest true positive rate.

The confusion matrix can also be analyzed. As it can be seen in Figure 4.8, the best results being again achieved by the vehicles class, the model correctly predicting 0.74 of the instances. 56% of the person instances were correctly predicted. Only 37% of the bicycle instances were detected. Also, the confusion matrix is in accordance to the low recall value, as it can be easily noticed from Figure 4.8 that a lot of instances failed to be detected, being classified as background (54% of the traffic sign instances, 50% of the person instances etc.).

Figure 4.9 presents the evolution of the box, object and class loss, as well as the evolution of the precision, recall and mAP during training. As it can be seen, the box validation loss is around 0.042. It can be noticed that there is a slight overfit after 150 epochs when analyzing the evolution of the object and class loss. However, the object loss is around 0.046 and the class loss is around 0.009.

Figure 4.7 pictures the F1-confidence curve. Ideally, a high value for both F1 score and confidence are desired. However, it can be seen that the F1 score decreases

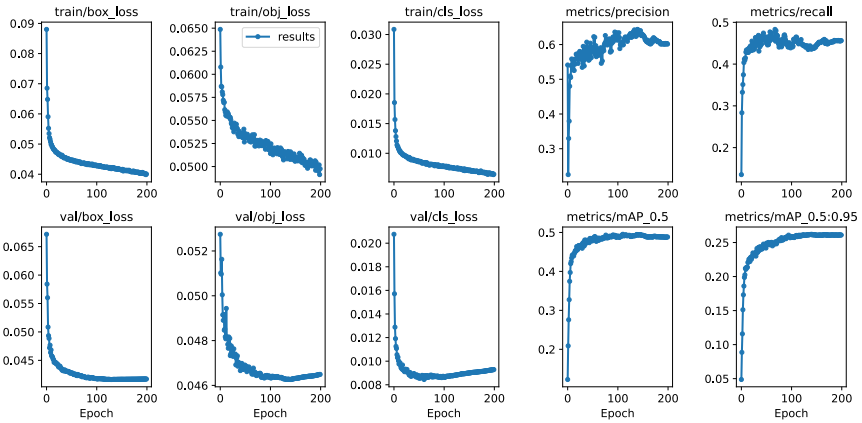


Figure 4.9 The evolution of box, object, class loss, as well as precision, recall, mAP50, mAP50-95 for training and validation when performing object detection on the original fisheye dataset

very much for all the classes after a confidence value of 0.6. Actually, except for the vehicles class, the decrease in the F1 score can be observed even after a confidence value of 0.5.

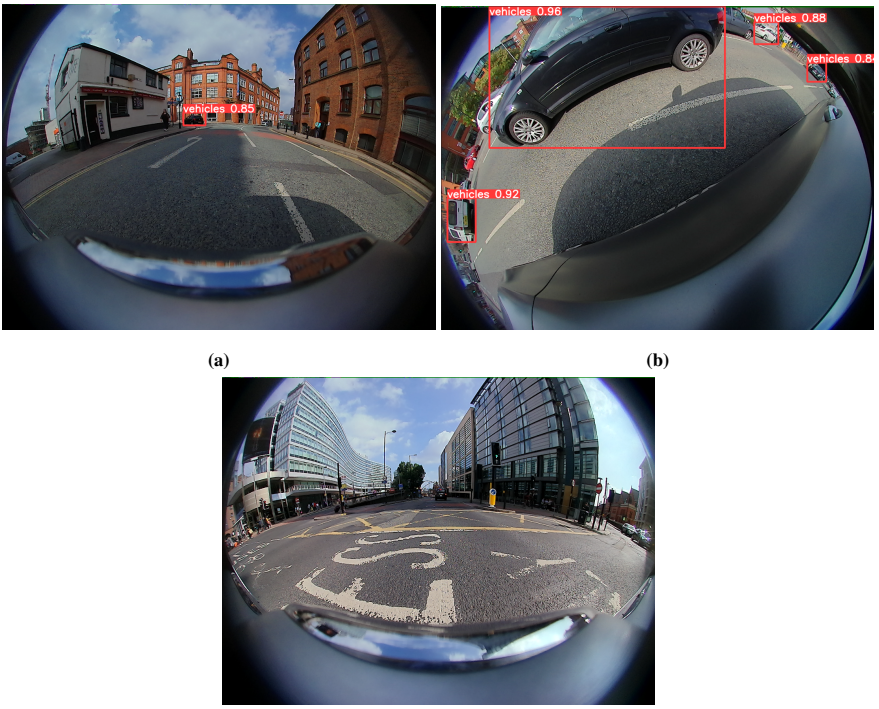
Last but not least, some examples of the detection results can be observed in Figure 4.10. In Figures 4.10a and 4.10b only some vehicles are detected (also notice that the person in Figure 4.10a fails to be detected), while in Figure 4.10c the model fails to detect anything.

We also considered the medium version of YOLO, but as it can be noticed in Listing 4.4, only the mAP was slightly improved.

```
190 epochs completed in 2.273 hours.
Model summary: 212 layers, 20869098 parameters, 0 gradients, 47.9 GFLOPs:

                 Class  Images  Instances      p      R  mAP50  mAP50-95
                   all     821       6953  0.582   0.54  0.533     0.303
              vehicles     821       4367  0.711  0.723  0.761      0.52
                person     821       1521  0.715  0.561  0.636     0.344
               bicycle     821        680  0.456   0.41  0.366     0.195
         traffic_light     821        137  0.592  0.526   0.46      0.21
          traffic_sign     821        248  0.437  0.482  0.443     0.245
```

Listing 4.4 Object detection validation results of the original fisheye dataset from WoodScape using YOLOv5 as the training model - medium model of YOLOv5. The picture shows for each class the number of images used for validation, the number of instances, precision (P), recall (R), mAP50 and mAP50-95 scores.

In addition, as it can be seen in Figure 4.11, the model was very overfitted. For instance, there is a considerable divergence between the training and the validation evolution of the object and class losses. For the validation, both object and class losses increase after approximately 80 epochs. Consequently, we only considered



(a) (b)

(c)

Figure 4.10 Example of detection result when using the original fisheye images. Notice the undetected vehicles in Figure 4.10b and the failure to detect anything in Figure 4.10c.

the results from the small YOLO model.

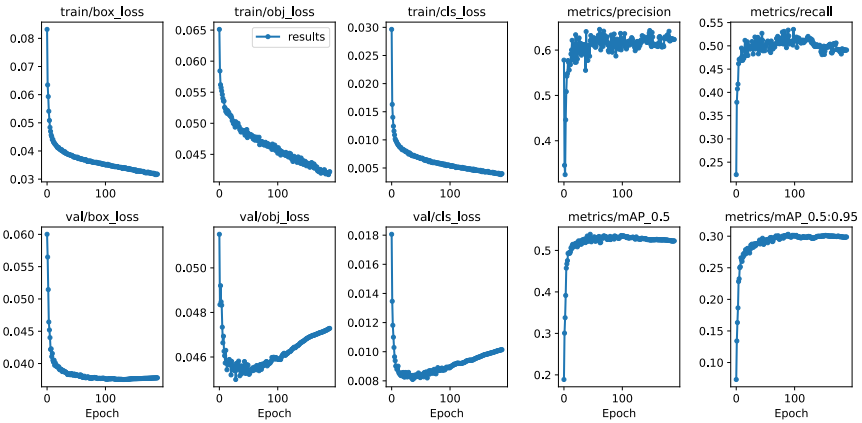


Figure 4.11 The evolution of box, object, class loss, as well as precision, recall, mAP50, mAP50-95 for training and validation when performing object detection on the original fish-eye dataset. The medium model of YOLOv5 was used. The divergence in loss metrics between the training and validation datasets serves as an indication of overfitting.

4.4 Cylindrical Rectification

The results for the cylindrical corrected images after training YOLO with them for 150 epochs can be seen in Listing 4.5. Similar to the previous baseline, we used the pretrained COCO weights for YOLO. The used batch size was 32. Overall, the precision with the cylindrical images is 0.578 and the recall is 0.46.

```
150  epochs  completed     in  1.599  hours  .
Model summary : 157 layers, 7023610 parameters, 0 gradients, 15 .8 GFLOPs:

              Class    Images    Instances         p         R    mAP50    mAP50-95
                all       819         7025     0.578      0.46    0.479       0.251
           vehicles       819         4374     0.663     0.714    0.743       0.472
             person       819         1532     0.654     0.533    0.567       0.287
            bicycle       819          679     0.481     0.339    0.343       0.168
     traffic_light        819          165     0.616      0.35    0.393       0.172
      traffic_sign        819          275     0.475     0.364    0.348       0.155
```

Listing 4.5 Object detection validation results of the cylindrical corrected dataset using YOLOv5 as the training model. The picture shows for each class the number of images used for validation, the number of instances, precision (P), recall (R), mAP50 and mAP50-95 scores.

The overall mAP50 is 0.479. Figure 4.12 presents the precision-recall curve. The results on the right correspond to the mAP50 results. The best result is for the vehicles, the most represented class, having the largest area under the curve and a mAP50 of 0.743. The lowest mAP50 is reached by the class bicycle (0.343).

The confusion matrix after running YOLO with cylindrical corrected images can be observed in Figure 4.14. The vehicles class has the best results, 75% of all

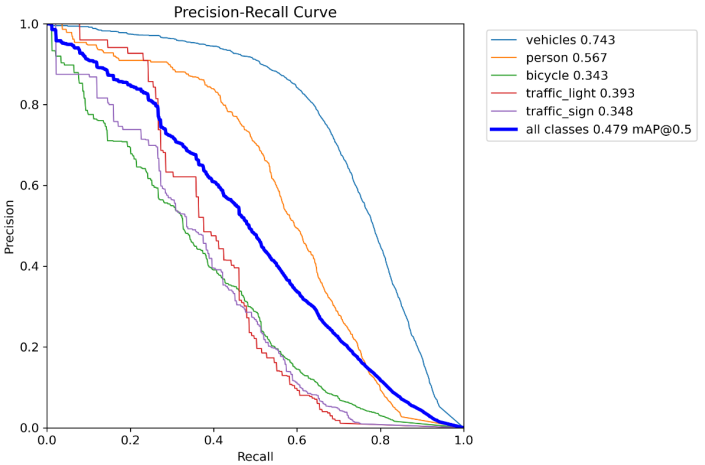


Figure 4.12 Precision-recall curve after performing object detection on the cylindrical corrected dataset. The results on the top right corner correspond to the mAP50 scores. The highest AP score is achieved by the vehicles class (light blue), while the bicycle class (green) performs the worst.

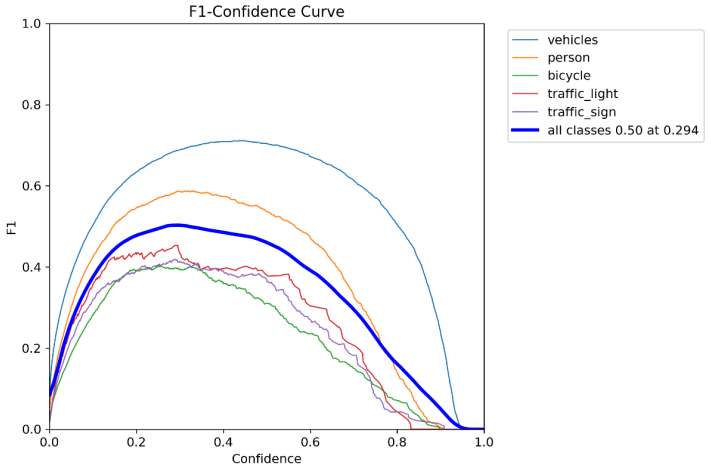


Figure 4.13 F1-confidence curve after performing object detection on the cylindrical corrected dataset. The vehicles (light blue) achieve the best F1 score, while the bicycle (green) and traffic sign (purple) classes perform the worst.

vehicles instances being correctly predicted. The worst results are for the bicycle class, only 35% of them being correctly identified. These results are in accordance to the low recall score, since as it can be noticed from Figure 4.14 a significant amount of objects are classified as background (59% of the traffic signs, 62% of the

traffic lights, etc.).

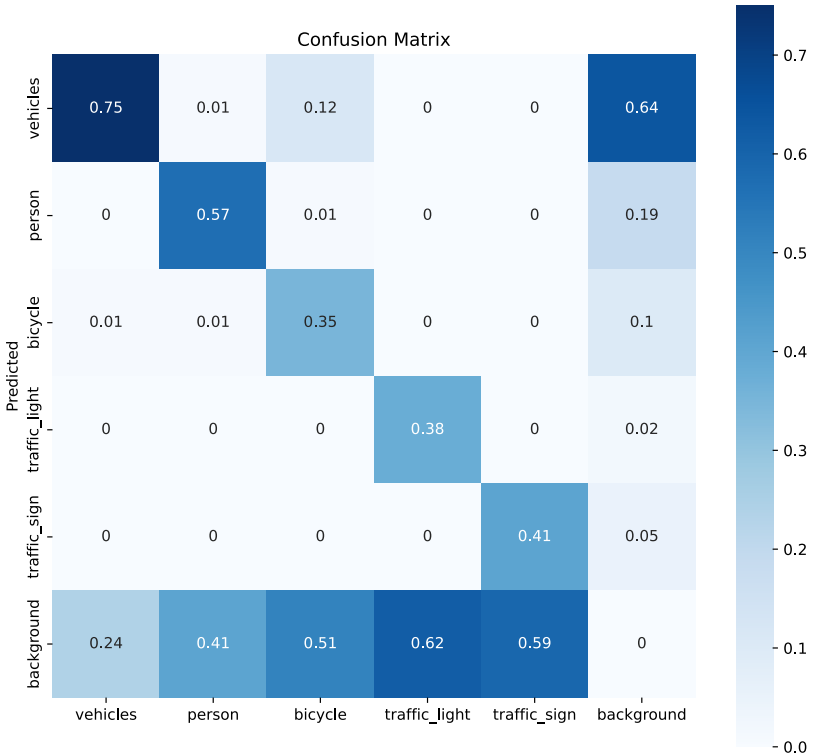


Figure 4.14 Confusion matrix after performing object detection on the cylindricaly corrected dataset. The class with the highest rate of true positives is the vehicles class, while the bicycle class has the lowest rate of true positives.

Figure 4.13 pictures the F1-confidence curve. The F1 score decreases very much after a confidence of 0.5 for all classes except the vehicles (which has a significant decrease in the F1 score after a confidence of 0.6).

Figure 4.15 presents the evolution of the box, object, and class loss, as well as the evolution of the precision, recall and mAP during training. The box validation loss is around 0.043, the object validation loss is around 0.041, and the validation class loss is around 0.009.

Some example of detection results can be seen in Figure 4.16. In Figure 4.16a most of the vehicles are detected (a few from the left side of the image are omitted). In Figure 4.16b the model is also good at detecting vehicles, but it detects just one person. In Figure 4.16c the model fails to detect any bicycle and also most of the vehicles.

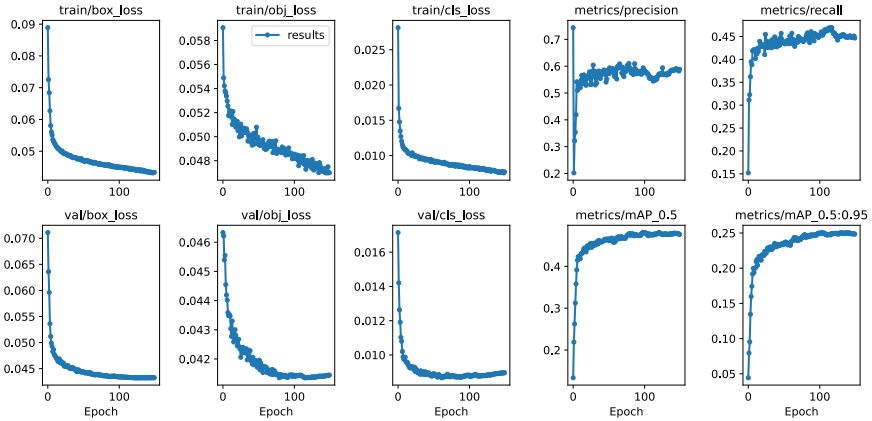
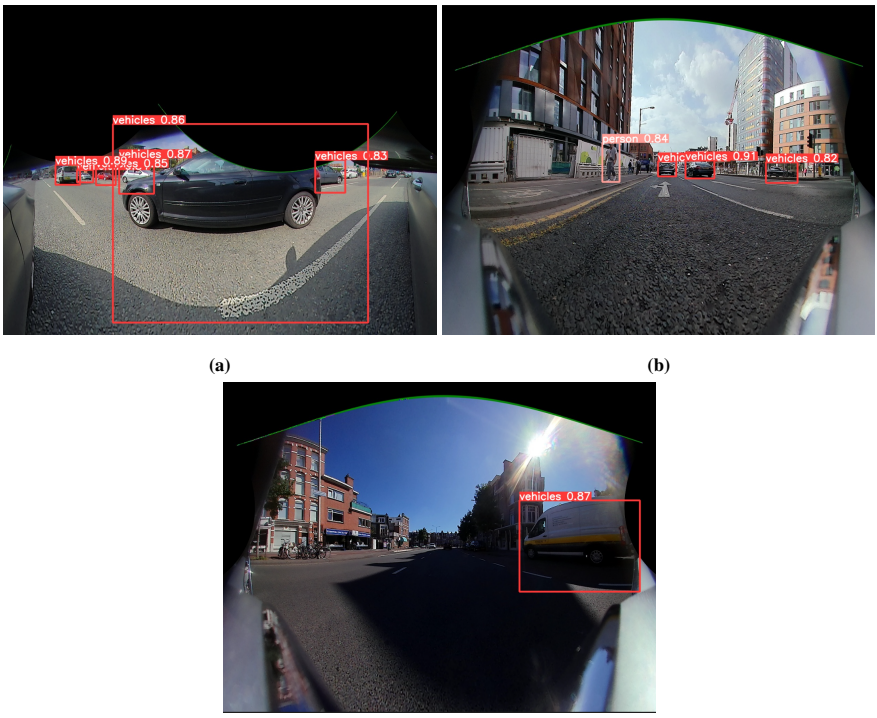


Figure 4.15 The evolution of box, object, class loss, as well as precision, recall, mAP50, mAP50-95 for training and validation when performing object detection on the cylindrical corrected dataset.



(a) (b)

(c)

Figure 4.16 Example of detection results when using cylindrical corrected images. Notice that there are a lot of detected vehicles in Figure 4.16a and Figure 4.16b, but the people and the bicycles are not detected. The bounding boxes are very large, covering more than just the object. This problem comes from converting the original fisheye annotations to cylindrical annotations.

4.5 Rectilinear Rectification

A first effect of the reduced field of view when doing the rectilinear correction is that there is also a reduction in the number of instances for the classes. This can be observed in Figure 4.17. When using the rectilinear images there are only 17500 vehicles instances, whereas when using the original fisheye images there are 35000 vehicles instances. However, the imbalance between the classes remains quite similar in both situations.

```
150 epochs completed in 1.415 hours.
Model summary: 157 layers, 7023610 parameters, 0 gradients, 15.8 GFLOPs:

                 Class  Images  Instances       p       R   mAP50  mAP50-95
                   all     819       3949   0.623   0.505   0.542     0.301
              vehicles     819       2336   0.764   0.707   0.776     0.497
                person     819        900   0.693   0.603   0.637     0.351
               bicycle     819        393   0.529   0.321   0.358     0.194
         traffic_light     819        124   0.645   0.452   0.502     0.244
          traffic_sign     819        196   0.486   0.444   0.438     0.218
```

Listing 4.6 Object detection validation results of the rectilinear corrected dataset from WoodScape using YOLOv5 as the training model. The picture shows for each class the number of images used for validation, the number of instances, precision (P), recall (R), mAP50 and mAP50-95 scores.

The results for the rectilinear corrected images after training YOLO with them for 150 epochs can be seen in Listing 4.6. Once again, the pretrained COCO weights for YOLO were used and the batch size was 32. The training took 1.415 hours. The overall recall is 0.505. The vehicles class has again the highest recall (0.707). The recall for some underrepresented classes improved: the recall for traffic light is 0.452 and the recall for traffic sign is 0.444.

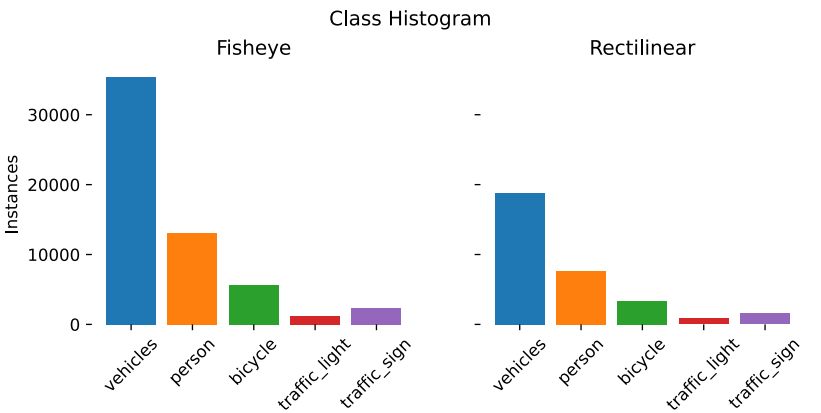


Figure 4.17 Comparison of the number of instances for each class for object detection when using the rectilinear corrected images (right) vs. when using the original fisheye images (left). Observe the drastic reduction in the number of instances due to cropping.

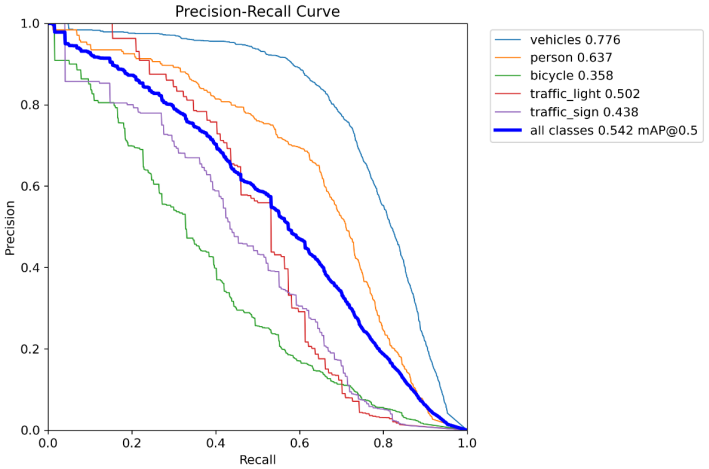


Figure 4.18 Precision-recall curve after performing object detection on the rectilinear corrected dataset. The results on the top right corner correspond to the mAP50 scores. The class with the highest AP is the vehicles class (light blue), while the class with lowest AP is the bicycle class (green).

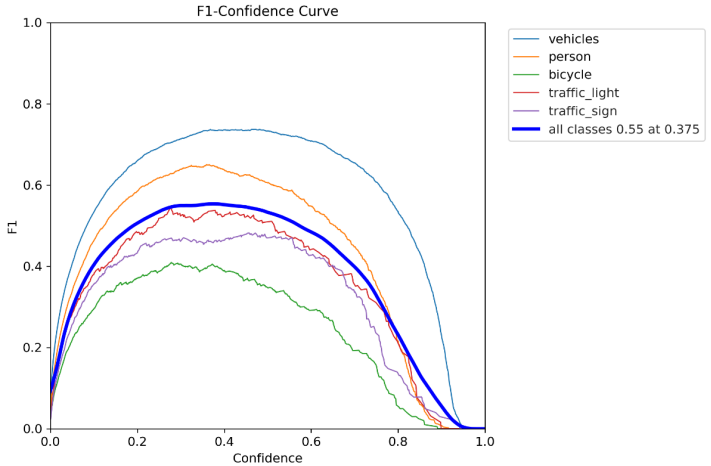


Figure 4.19 F1-confidence curve after performing object detection on the rectilinear corrected dataset. The class with the highest F1 score is the vehicles class (light blue), while the class with the lowest F1 score is the bicycle class (green).

The overall mAP50 with rectilinear corrected images is 0.542. Figure 4.18 presents the precision-recall curve. The results on the right side correspond to the mAP50 results. The best result is for the vehicles, having the largest area under

the curve and a mAP50 of 0.776. The class person has also a reasonable mAP50 now (0.637). The lowest mAP50 is reached by the class traffic sign, with a score of 0.438.

The confusion matrix after running YOLO with rectilinear corrected images can be observed in Figure 4.20. The vehicles class has the best results, 78% of all vehicles instances being correctly predicted. The worst results are for the bicycle class, only 40% of them being correctly identified. These results are in accordance to the recall score, since as it can be noticed from Figure 4.20 a significant amount of objects are classified as background (46% of the traffic signs, 47% of the traffic lights, etc.).

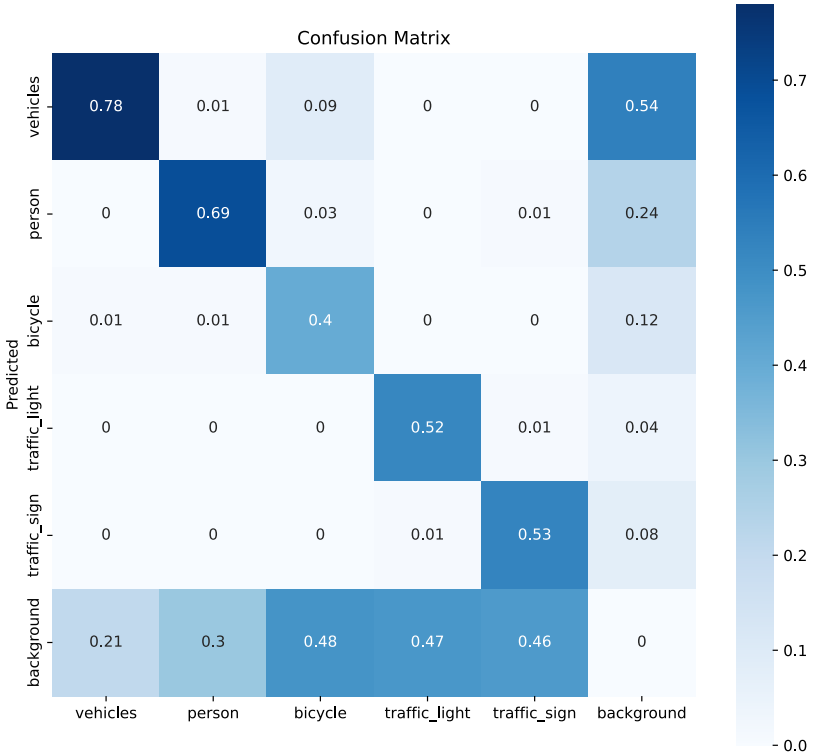


Figure 4.20 Confusion matrix after performing object detection on the rectilinear corrected dataset. The vehicles class has the highest true positive rate, while the bicycle class the lowest true positive rate.

Figure 4.19 pictures the F1-confidence curve. The F1 score decreases very much after a confidence of 0.6 for all classes except the vehicles (that has a significant decrease in the F1 score after a confidence of 0.7).

Figure 4.21 presents the evolution of the box, object, and class loss, as well as

the evolution of the precision, recall and mAP during training. The box validation loss is around 0.040, the object validation loss is around 0.028, and the validation class loss is around 0.010.

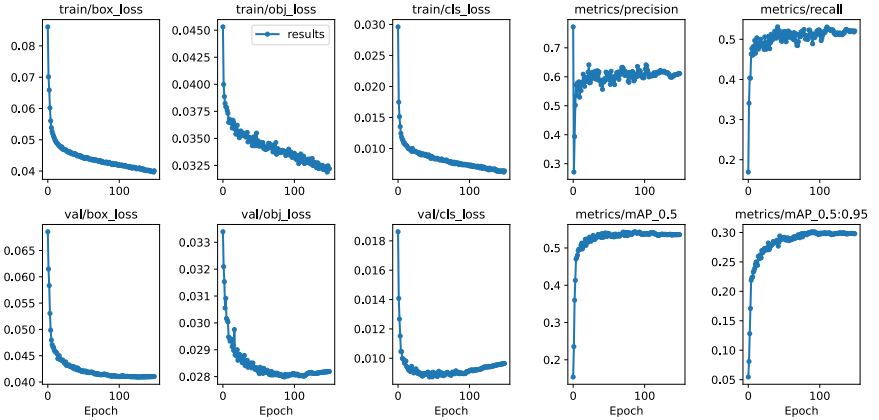
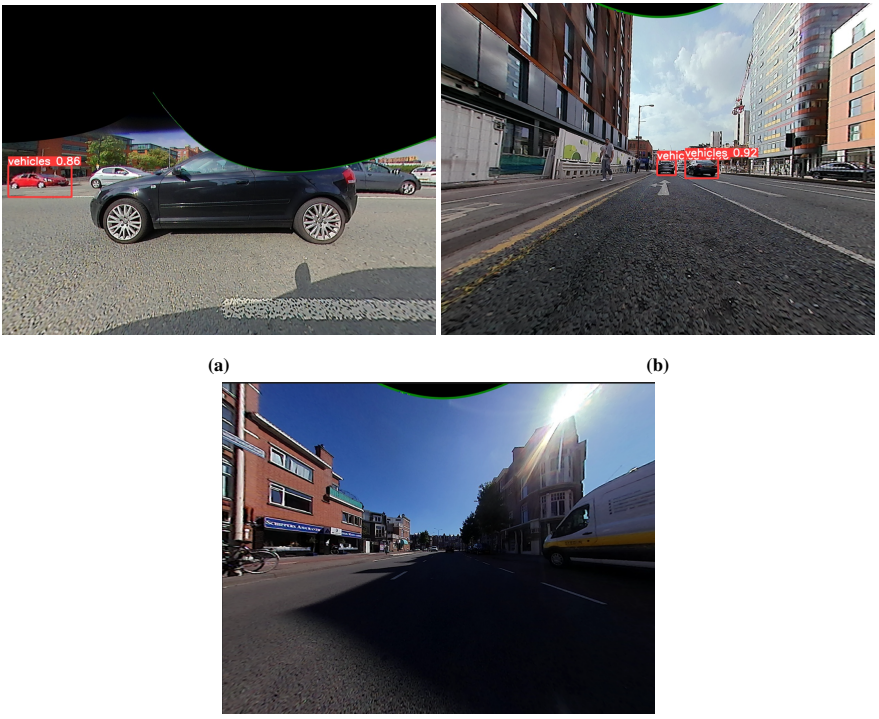


Figure 4.21 The evolution of box, object, class loss, as well as precision, recall, mAP50, mAP50-95 for training and validation when performing object detection on the rectilinear corrected dataset

The amount of information lost can be observed also in the examples of detection results from Figure 4.22 (compared to the images in Figure 4.16). For example, in Figure 4.22a there is only one car instance detected, and there are only 4 car instances in total because of the reduction in the field of view.



(a) (b)

(c)

Figure 4.22 Example of detection results when using rectilinear corrected images. Notice that there are not so many objects in the pictures due to the reduction in the field of view. Moreover, many vehicles fail to be detected in Figure 4.22a. The bounding boxes are very large, covering more than just the object. This problem comes from converting the original fisheye annotations to rectilinear annotations.

5

Semantic Segmentation

This chapter covers the baseline and rectification experiments for semantic segmentation. The input for the baseline consists of the fisheye distorted images from the WoodScape dataset [1] and their corresponding semantic segmentation annotations. The model used to perform the training is YOLOv5 again [21].

Like with the object detection experiments, rectified versions of the WoodScape dataset were used to train YOLOv5 and test whether such methods improved detection results.

5.1 Pre-Processing the Annotations

WoodScape provided instance annotations in the below form:

```
{"00000_FV.json": {"annotation": [{"id":
"C0E39A62-5569-2062-F133-ABBCF8A97C3E",
"segmentation": [[193.0, 3.3810000000000002],
[246.66666666666669, 3.3810000000000002],
[595.6666666666667, 3.3810000000000002],
[1102.269592476489, 3.3810000000000002],
[1280.0, 3.3810000000000002], [1280.0, 0.0],
[0.0, 0.0], [0.0, 3.3810000000000002]],
"states": {}, "tags": ["green_strip"],
"z_order": 0}, {"id":
"C1EC99BB-ED0B-4DF0-297F-3C3F1EF7A916",
"segmentation": [[1280.0, 962.619],
[1280.0, 966.0], [0.0, 966.0],
[0.0, 962.619]], "states": {},
"tags": ["green_strip"], "z_order": 0}...
```

Listing 5.1 A snippet of a JSON file which contains information about all the segmentations in a single scene. There is additional information here which will not be used such as *states*, *id* and *z_order*.

As it can be noticed, the annotations do not match COCO format. Therefore, we modified a script that was reading all these annotations and was mapping the tags to 9 classes (0 - road, 1 -lanemarks, 2 - curb, 3 - person, 4 - rider, 5 - vehicles, 6 -

bicycle, 7 - motorcycle, 8 - traffic_sign) and we converted the given format to COCO format for semantic segmentation. After processing, the segmentation annotations have the following format:

```
3 0.94437210 0.47182923 0.94220196 0.47710106 ...
3 0.28647534 0.36813700 0.28011588 0.36939577 ...
1 0.30590570 0.42927810 0.32862750 0.42189023 ...
```

Listing 5.2 Example of a semantic segmentation annotation file for WoodScape after processing the JSON file. In this snipped there are three polygons, one for each new line. The information is space-delimited with the first number defining the polygons class and the following pairs defining (x, y) points of the polygon.

As it can be noticed in Listing 5.2, the information from the annotation file has the following structure: class ID code followed by a set of (x, y) coordinates for the segmentation mask. The coordinates are normalized with respect to the width and height of the image.

Similar to the method that we used for object detection, the generation of the mask coordinates is also needed for the cylindrical and rectilinear images. An example of an annotated fisheye image with the masks coordinates for segmentation can be observed in Figure 5.1a.

We used the segmentation annotation files for the fisheye images and performed the 2D to 3D projection for the coordinates of the masks in order to get the real-world space coordinates. The real-world space coordinates were then used to perform a cylindrical/rectilinear 3D to 2D projection in order to get the corresponding mask coordinates for the corresponding cylindrical/rectilinear corrected images. An example of an annotated cylindrical image can be seen in Figure 5.1b and an example of an annotated rectilinear image can be seen in Figure 5.1c.

(a) (b) (c)

Figure 5.1 Example of an annotated (a) - fisheye, (b) - cylindrical, (c) - rectilinear image from the WoodScape dataset

5.2 Original Dataset

We used the original fisheye images from WoodScape and their corresponding annotations for semantic segmentation as input for YOLO. We initialized the weights with the ones from COCO. We trained the model for 100 epochs using a batch size of 16. The reason why we chose a batch size of 16 instead of a batch size of 32 is the computation capacity limit. The AWS virtual machine was unable to perform segmentation training with a batch size of 32. As it can be observed in Listing 5.3, the training took 4.412 hours. The validation results are also displayed in Listing 5.3.

```
100 epochs completed in 4.412 hours.
Model "summary: 165 layers, 7419998 parameters, 0 gradients, 25.8 GFLOPs:

       Class  Images  Instances    Box(P       R    mAP50  mAP50-95)  Mask(P       R    mAP50  mAP50-95)
         all     821       3826    0.606   0.355   0.388      0.208   0.533   0.282   0.296      0.131
        road     821       5360    0.677   0.284   0.354      0.225   0.565   0.207   0.231      0.166
  lanemarks      821      16534    0.578   0.413   0.443      0.249   0.468   0.301    0.31       0.14
        curb     821       3401    0.623   0.342   0.378      0.218   0.409   0.201   0.187     0.0607
      person     821       2316    0.642   0.274   0.333      0.159    0.56    0.21   0.246     0.0933
       rider     821        710     0.7    0.321   0.383       0.16   0.595   0.238   0.266     0.0679
    vehicles     821       6789    0.716   0.616   0.666      0.418   0.714   0.565   0.625      0.349
     bicycle     821       1895    0.461    0.29   0.281      0.128   0.445   0.244   0.228     0.0798
  motorcycle     821        606    0.623   0.299   0.344       0.17   0.639   0.268   0.301      0.116
traffic_sign     821        654    0.435   0.355   0.309      0.143   0.407     0.3    0.27      0.102
```

Listing 5.3 Semantic segmentation validation results of the original fisheye dataset from WoodScape using YOLOv5 as the training model. The picture shows for each class the number of images used for validation, the number of instances, precision (P), recall (R), mAP50 and mAP50-95 scores for both the boxes and the segmentation masks.

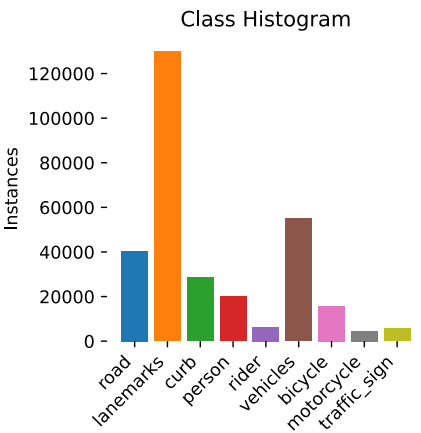


Figure 5.2 Distribution of class instances for semantic segmentation in the original fisheye images from WoodScape. The most represented class is the lanemraks class, followed by the vehicles class. At the opposite pole, there is the motorcycle class.

As it can be noticed in Listing 5.3, there are two sets of metrics: one is referring to the bounding boxes (this one was also used in our previous results, when doing

only object detection) and the other one is referring to the segmentation mask. For the bounding box results, the overall precision is 0.606, the recall is lower at 0.355 (meaning that a significant number of the objects are not detected), and the mAP50 is 0.388. As for the masks, the overall precision is 0.533, the recall is 0.282, and the mAP50 is 0.296. Also, as it can be seen in Figure 5.2, the dataset is imbalanced, having significantly more instances of lanemarks and vehicles than the other classes. The least represented classes are the motorcycle, rider, and traffic_sign.

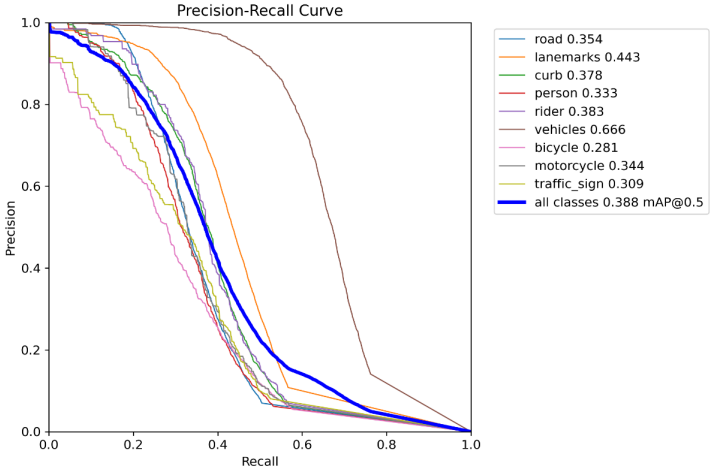
Figure 5.3a provides a plot of the precision-recall curve for the bounding boxes. The results on the right correspond to the class mAP50. The best results are achieved by the vehicles (0.666) and the lanemarks (0.443) classes. The worst results are obtained by the bicycle (0.282), traffic_sign (0.309), and person (0.333) classes. Figure 5.3b illustrates the precision-recall curve for the masks, having the corresponding mAP50 results on the right side. The best results are again achieved by the vehicles (0.625) and the lanemarks (0.310) classes. The vehicles have the largest area under the curve for both boxes and masks. The worst results for the masks are obtained for the curb (0.187), bicycle (0.228), and road (0.231) classes.

The confusion matrix can be observed in Figure 5.4. The vehicles class achieves again the best results, 62% of the vehicles being correctly identified. The lanmearks class has the second best results, even though 56% of the instances are classified as background. The rest of the classes have even worse results. For example, 68% of the road and person instances failed to be detected. However, these results are in accordance to the low values for recall.

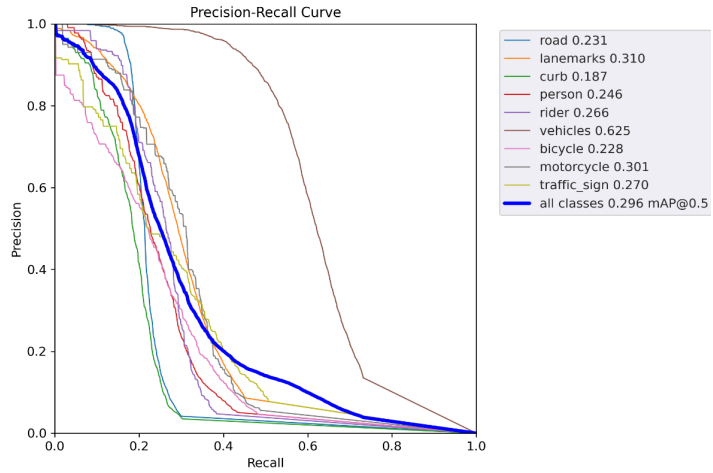
Figure 5.5a presents the F1 curve for the bounding boxes. As expected, the vehicles achieve again the best performance, the F1 score significantly decreasing after a confidence score of approximately 0.5. Overall, for all classes, the F1 score decreases after a confidence score of 0.277. Similarly, Figure 5.5b presents the F1 curve for the masks. The F1 score for the vehicles significantly decreases after a confidence score of 0.55, while the F1 score for all classes decreases a lot after a confidence score of 0.295.

Figure 5.6 presents the training and validation results, illustrating the evolution of the box, segmentation, object, and class losses, as well as the evolution of the precision, recall, and mAP50 for both the bounding boxes and the masks. In general, the performances stop improving after around 87 epochs. The box loss is around 0.083, the segmentation loss is around 0.0573, the object loss is around 0.107, and the class loss is approximately 0.0225.

Some segmentation examples can be seen in Figure 5.7. In Figure 5.7a, the model fails to detect the bus and some lanemarks, but it is pretty good at detecting the other objects. In Figure 5.7b the model misses some vehicles from the left corner, where the distortion is stronger. Some strongly distorted objects are also missed from Figure 5.7c.



(a)



(b)

Figure 5.3 Precision-recall curve after performing semantic segmentation on the original fisheye dataset: (a) for the bounding boxes, (b) for the segmentation masks. The results in the top right corner correspond to the mAP50 scores. For both the boxes and masks, the vehicles (brown) achieve the best AP sore. The bicycle class (pink) has the lowest AP for the bounding boxes, while the curb (green) and the road (blue) classes have the lowest AP for the segmentation masks.

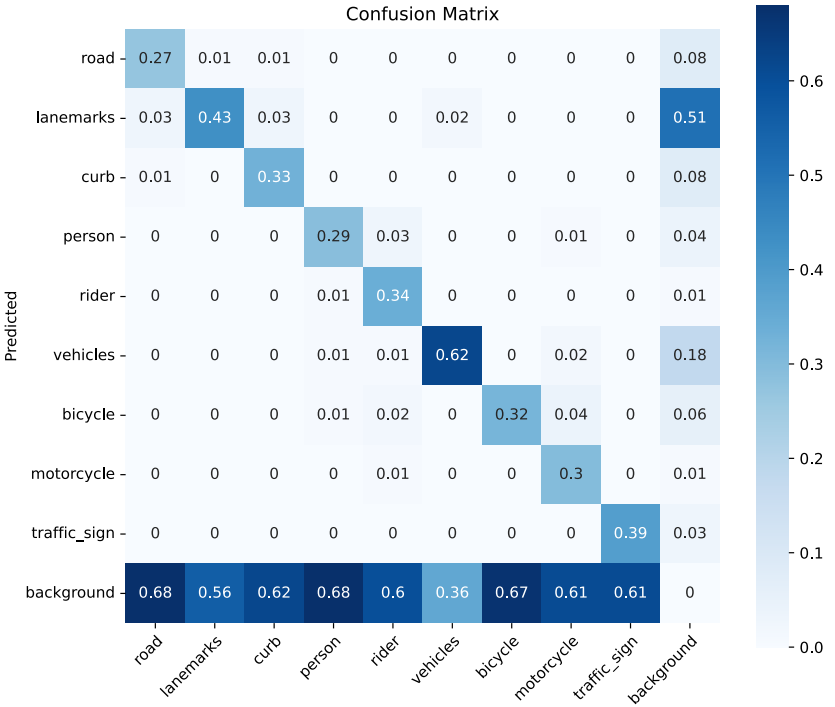
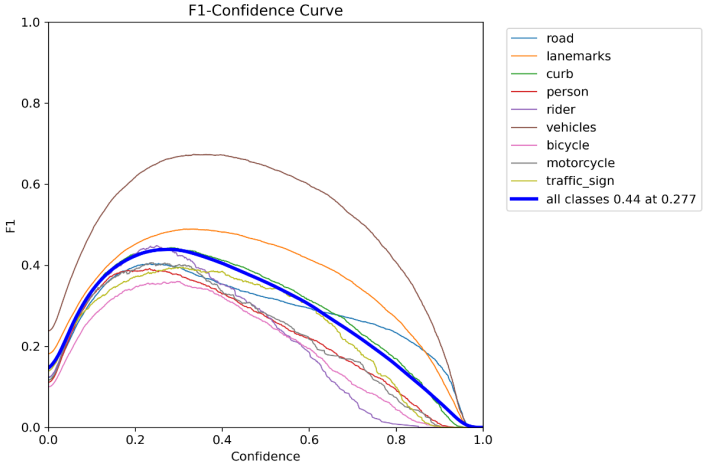
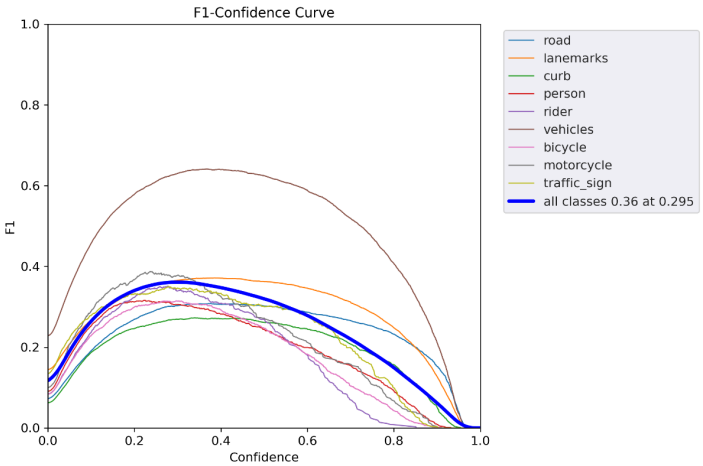


Figure 5.4 Confusion matrix after performing semantic segmentation on the original fisheye dataset. The vehicles have the highest true positive rate, while the road has the lowest true positive rate.



(a)



(b)

Figure 5.5 F1-confidence curve after performing semantic segmentation on the original fish-eye dataset: (a) for the bounding boxes, (b) for the segmentation masks. The vehicles (brown) have the highest F1 score for both the boxes and the masks. The bicycle class (pink) has the lowest F1 score for the bounding boxes, while the curb (green) has the lowest F1 score for the segmentation masks.

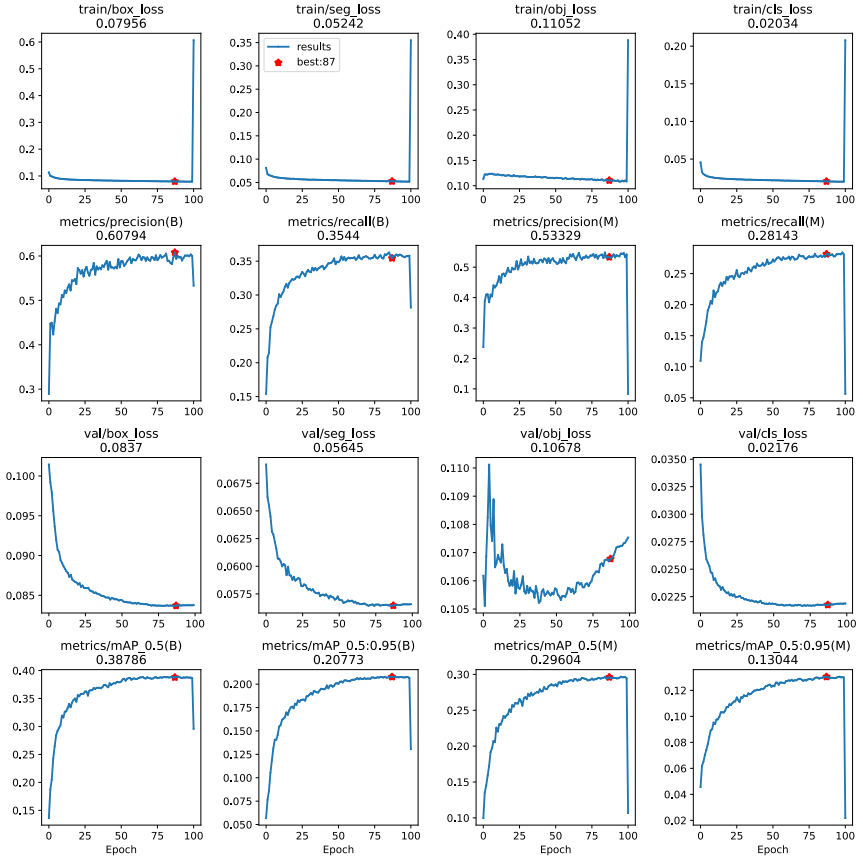


Figure 5.6 The evolution of box, segmentation, object, class loss, as well as precision, recall, mAP50, mAP50-95 for training and validation when performing semantic segmentation on the original fisheye dataset

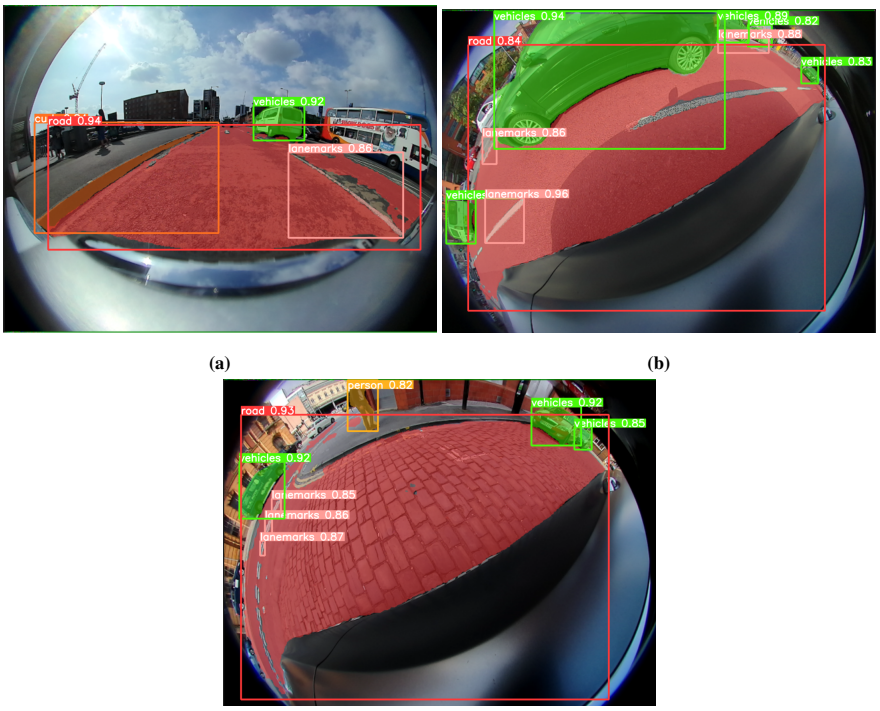


Figure 5.7 Example of segmentation result when using the original fisheye images. Most objects are detected, but it fails to detect some objects in the corners, where the distortion is more visible. Notice that the bounding boxes do not fit for every object (e.g. the bounding box for the road covers almost the whole picture).

5.3 Cylindrical Rectification

The segmentation results when using the cylindrical corrected images can be observed in Listing 5.4. We considered again 100 epochs and a batch size of 16. The COCO weights were used for initialization. As it can be noticed in Listing 5.4, the training took 3.839 hours. The overall results for the bounding boxes are the following: a precision of 0.559, a recall of 0.308, and a mAP50 of 0.332. The overall results for the masks are the following: a precision of 0.5, a recall of 0.246, and a mAP50 of 0.26.

```
100 epochs completed in 3.839 hours.
Model summary: 165 layers, 7419998 parameters, 0 gradients, 25.8 GFLOPs:

       Class  Images  Instances  Box(P       R   mAP50  mAP50-95)  Mask(P       R   mAP50  mAP50-95)
         all     798      36830  0.559   0.308   0.332      0.167     0.5   0.246    0.26       0.11
        road     798       5156  0.619   0.245   0.292      0.182   0.591    0.21   0.243      0.136
   lanemarks     798      15759  0.549   0.323   0.348       0.18   0.435   0.236   0.249      0.114
        curb     798       3316  0.595   0.275   0.297      0.149   0.505   0.211    0.22     0.0826
      person     798       2248  0.561    0.22   0.257      0.102   0.546   0.183    0.22     0.0795
       rider     798        675  0.608   0.241   0.308      0.119   0.478   0.156   0.179     0.0444
    vehicles     798       6633   0.68   0.614   0.654      0.405   0.627   0.531   0.572      0.304
     bicycle     798       1832  0.443   0.264   0.258      0.114   0.418   0.218   0.204      0.071
  motorcycle     798        576   0.59   0.262   0.315      0.152   0.601   0.236   0.273      0.105
traffic_sign     798        635  0.387   0.324   0.263      0.101   0.294   0.231   0.176     0.0563
```

Listing 5.4 Semantic segmentation validation results of the cylindrical corrected dataset using YOLOv5 as the training model. The picture shows for each class the number of images used for validation, the number of instances, precision (P), recall (R), mAP50 and mAP50-95 scores for both the bounding boxes and the segmentation masks.

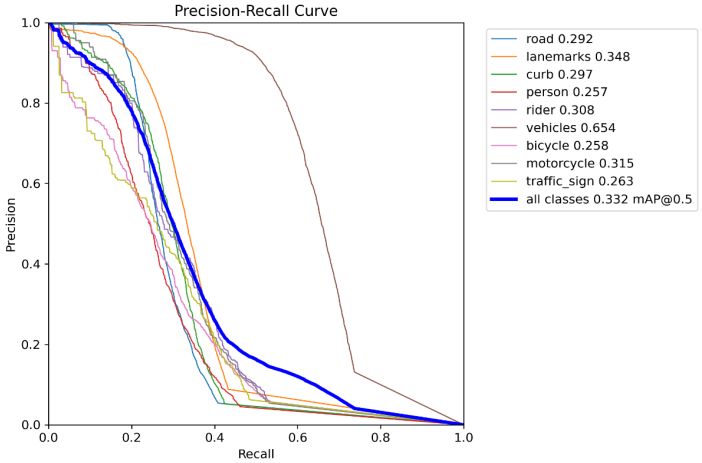
Figure 5.8a presents the precision-recall curve for the bounding boxes with the corresponding mAP50 results (right side). The vehicles have the largest area under the curve, with a mAP50 of 0.654. The worst results are obtained by the person (0.257), the bicycle (0.258), and the traffic_sign (0.263) classes. As for the mask results, they can be seen in Figure 5.8b. The vehicles have again the highest score (0.572). The classes with the lowest score are: traffic_sign (0.176) and rider (0.179).

The confusion matrix can be found in Figure 5.9. The best performances are achieved by the vehicles class, 60% of them being correctly classified. The worst results are obtained by the person (only 22% of them are correctly identified), the rider (23% of the instances are correctly classified), and the road (24% are classified accordingly) classes.

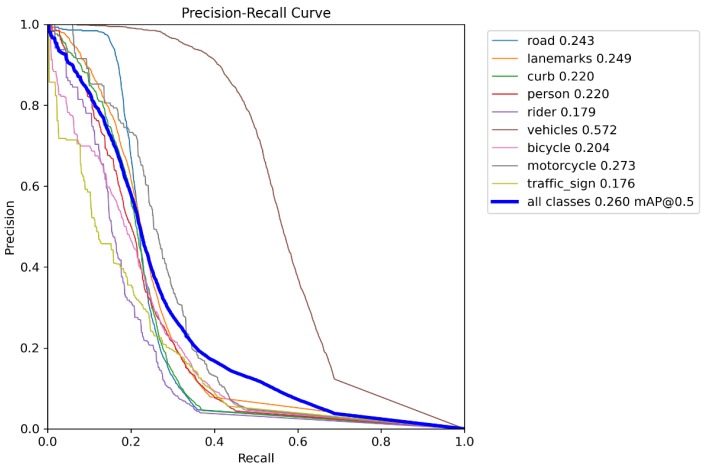
The F1 score for the bounding boxes can be seen in Figure 5.10a. The vehicles have the best results, the F1 score quickly decreasing after a confidence score above 0.5. The F1 score decreases after a confidence score above 0.241. The F1 score for the masks can be observed in Figure 5.10b. Once again, the vehicles have the best results. The F1 score for the vehicles decreases after a confidence score of above 0.55, but the F1 score for all classes significantly decreases after a confidence score of above 0.255.

The training and validation results can be analyzed in Figure 5.11. The validation box loss is around 0.095, the segmentation loss is approximately 0.0623, the object loss is near 0.070, and the class loss is 0.022.

Some examples of segmentation results can be seen in Figure 5.12. The results



(a)



(b)

Figure 5.8 Precision-recall curve after performing semantic segmentation on the cylindrical corrected dataset: (a) for the bounding boxes, (b) for the segmentation masks. The results on the top right corner correspond to the mAP50 scores. The vehicles class (brown) achieves the best AP score for both the bounding boxes and the segmentation masks. The lowest APs for the bounding boxes are obtained by the person class (red) and the bicycle (pink) classes. The lowest APs for the segmentation masks are obtained by the traffic sign (light green) and the rider (purple) classes.

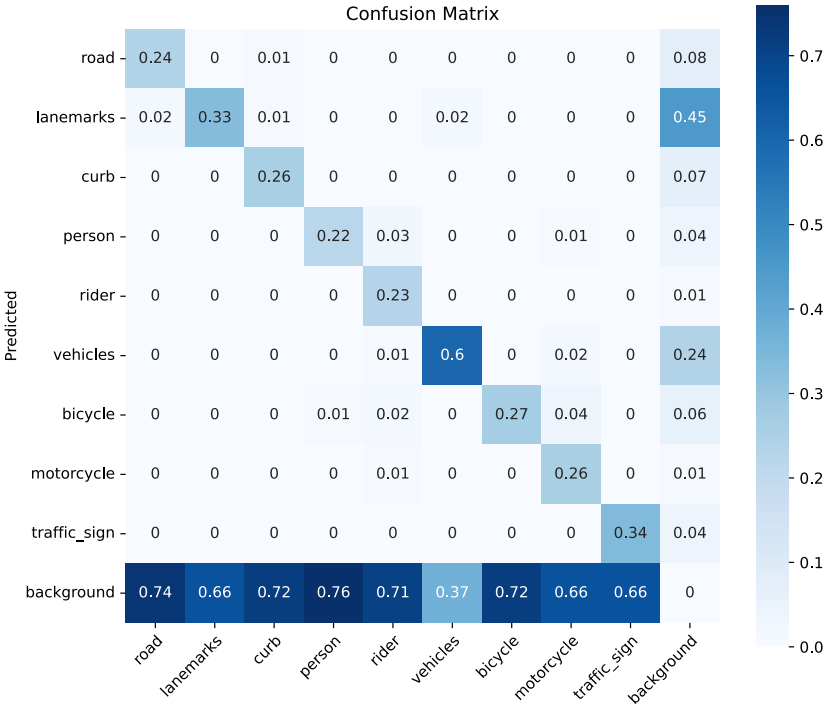
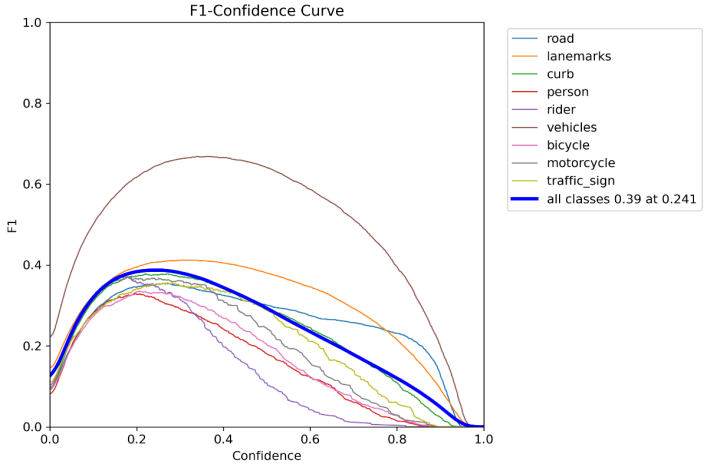
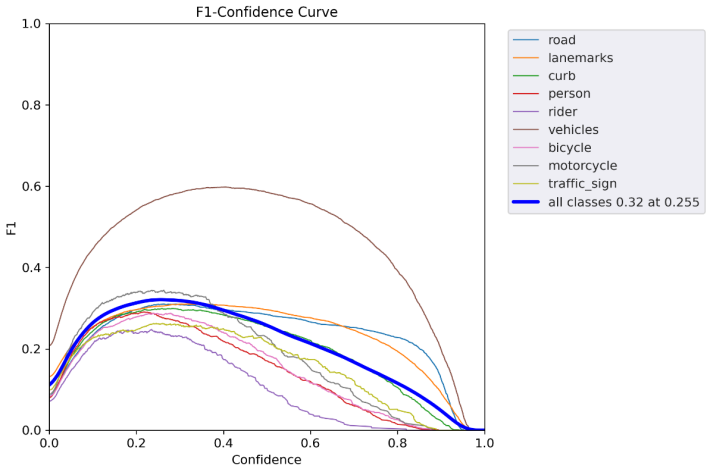


Figure 5.9 Confusion matrix after performing semantic segmentation on the cylindrical corrected dataset. The highest true positive rate is achieved by the vehicles class, while the class with the lowest true positive rate is the person class.

from Figures 5.12a and 5.12b are very similar to the results from Figures 5.7a and 5.7b. Some lanemarks failed to be identified in Figure 5.12c (compared to Figure 5.7c).



(a)



(b)

Figure 5.10 F1-confidence curve after performing semantic segmentation on the cylindrical corrected dataset: (a) for the bounding boxes, (b) for the segmentation masks. The class with the highest F1 score for both the boxes and the masks is the vehicles class (brown). The person class (red) has the lowest F1 score for the bounding boxes, while the rider class (purple) has the lowest F1 score for the masks.

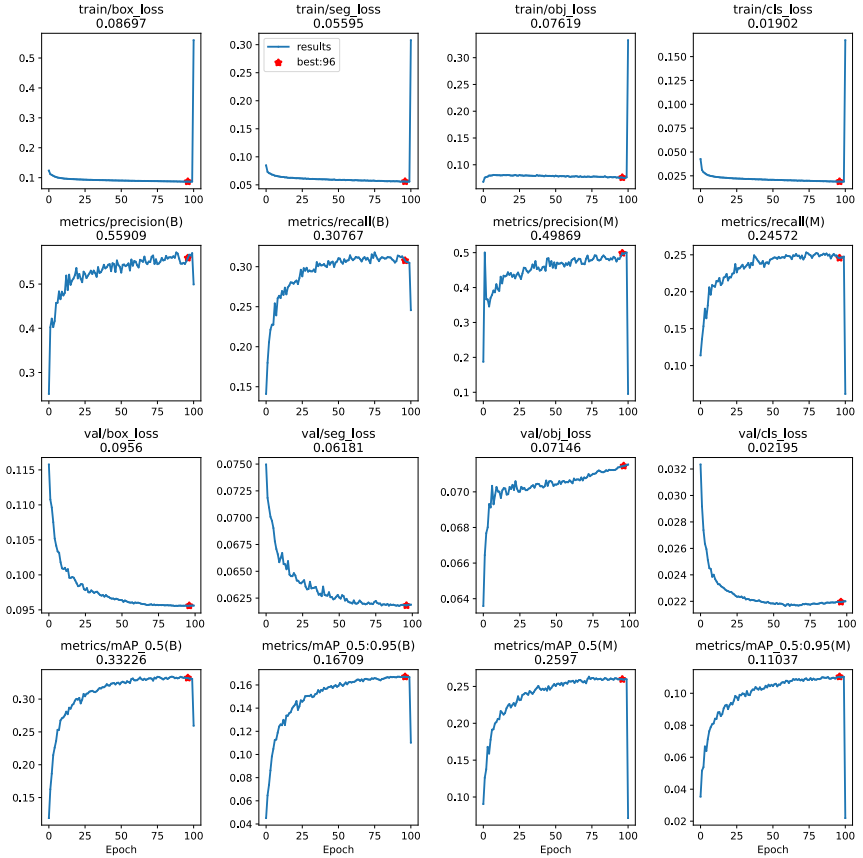
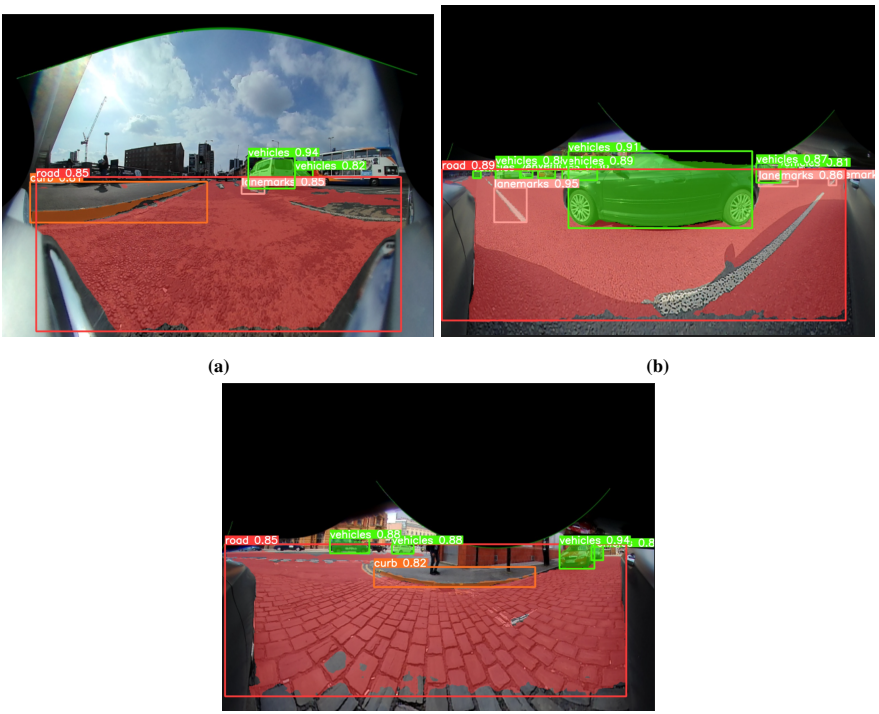


Figure 5.11 The evolution of box, segmentation, object, class loss, as well as precision, recall, mAP50, mAP50-95 for both the bounding boxes and segmentation masks for training and validation when performing semantic segmentation on the cylindrical corrected dataset



(a) **(b)**

(c)

Figure 5.12 Example of segmentation result when using the cylindrical corrected images. Most of the objects are detected. Notice that the bounding boxes are tighter than the bounding boxes obtained from doing object detection.

5.4 Rectilinear Rectification

Listing 5.5 presents a summary of the segmentation results with YOLO when having as input the rectilinear corrected images. The model was trained for 100 epochs, using a batch size of 16 and COCO weights for initialization. As it can be observed, the training took 3.504 hours. The boxes mAP50 for all classes is 0.392, while the masks mAP50 is 0.315. The box precision for all classes is 0.604 and the box recall for all classes is 0.37. The mask precision for all classes is 0.552 and the mask recall for all classes is 0.303.

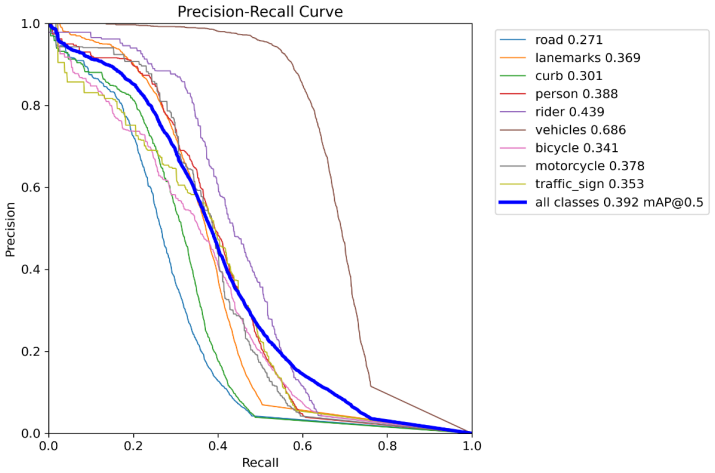
```
100 epochs completed in 3.504 hours.
Model summary: 165 layers, 7419998 parameters, 0 gradients, 25.8 GFLOPs

      Class  Images  Instances  Box(P       R  mAP50  mAP50-95)  Mask(P       R  mAP50  mAP50-95)
        all     823      25239  0.604    0.37  0.392      0.199   0.552   0.303  0.315      0.13
       road     823       3759  0.577   0.247  0.271      0.119   0.405   0.147  0.126    0.0423
  lanemarks     823      10237  0.575    0.35  0.369      0.182   0.464    0.25  0.253     0.108
       curb     823       2588  0.613   0.281  0.301      0.129   0.527   0.209  0.213    0.0648
     person     823       1622  0.627   0.352  0.388      0.186   0.635   0.317  0.352     0.143
      rider     823        451  0.646   0.384  0.439      0.203   0.644   0.322  0.345     0.103
   vehicles     823       4356  0.749   0.637  0.686      0.443   0.729   0.575  0.629     0.342
    bicycle     823       1315  0.521   0.345  0.341      0.171   0.523   0.301  0.311     0.121
 motorcycle     823        411  0.641    0.34  0.378      0.197   0.595   0.278  0.322     0.145
traffic_sign     823        500  0.484   0.398  0.353      0.158    0.45   0.328  0.286     0.104
```

Listing 5.5 Semantic segmentation validation results of the rectilinear corrected dataset using YOLOv5 as the training model. The picture shows for each class the number of images used for validation, the number of instances, precision (P), recall (R), mAP50 and mAP50-95 scores for both the bounding boxes and the segmentation masks.

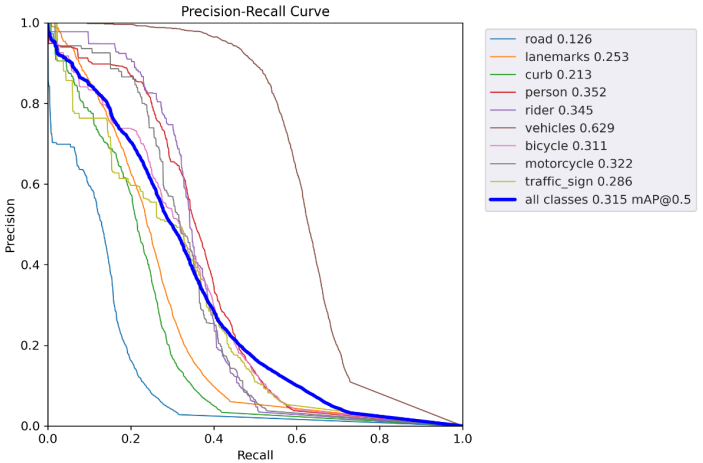
Figure 5.13 Distribution of class instances for semantic segmentation in the rectilinear corrected images (right) compared to the distribution of class instances in the original fisheye images (left). Observe the reduction of the number of instances caused by cropping (compared to the number of instances in Figure 5.2).

The drastic reduction in the number of objects due to cropping the images can be noticed in Figure 5.13. While the original dataset contained 120000 instances for lanemarks and 60000 instances for vehicles (see Figure 5.2), the rectilinear images

have only 80000 instances of lanemarks and 40000 instances of vehicles.



(a)



(b)

Figure 5.14 Precision-recall curve after performing semantic segmentation on the rectilinear corrected dataset: (a) for the bounding boxes, (b) for the segmentation masks. The results on the top right corner correspond to the mAP50 scores. The vehicles (brown) perform the best for both boxes and masks, while the road class (blue) performs the worse.

The precision-recall curve for the bounding boxes is illustrated in Figure 5.14a, having the corresponding class mAP50 on the right side. The vehicles achieved the best results, having a mAP50 of 0.686. The worst results are obtained by: road with

a mAP50 of 0.271 and curb, having a mAP50 of 0.301. Figure 5.14b presents the precision-recall curve for the masks, with their corresponding class mAP50 on the right side. The vehicles have the best performance, with a mAP50 of 0.629. The classes with the poorest mAP50 are: road with 0.126 and curb with 0.213.

The confusion matrix can be analyzed in Figure 5.15. 63% of the vehicles are correctly classified. At the opposite side, only 26% of the road instances and 29% of the curbs are correctly classified.

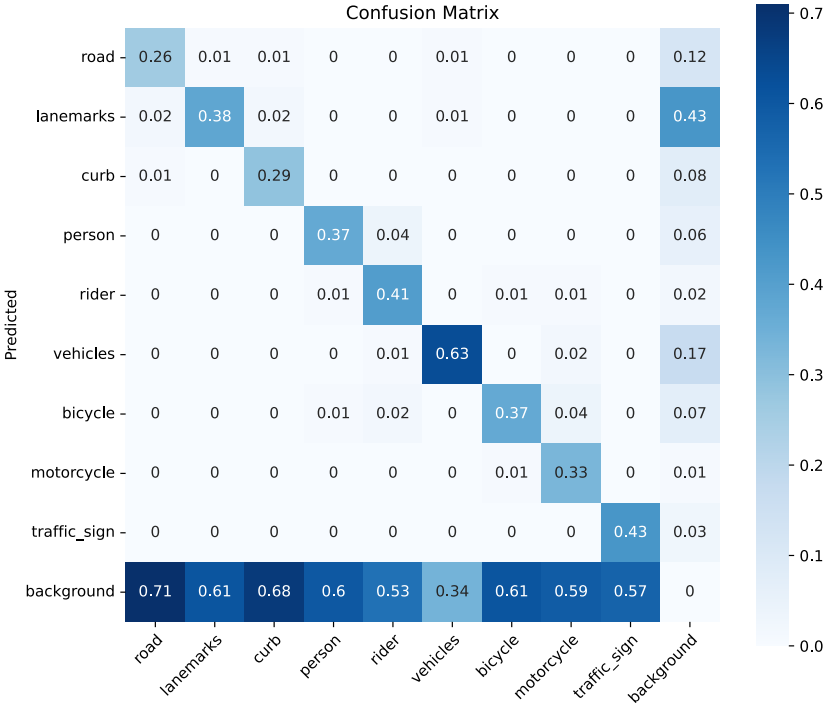
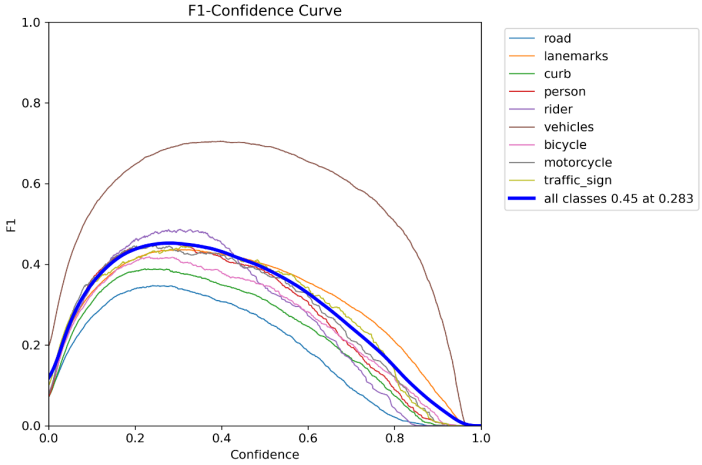


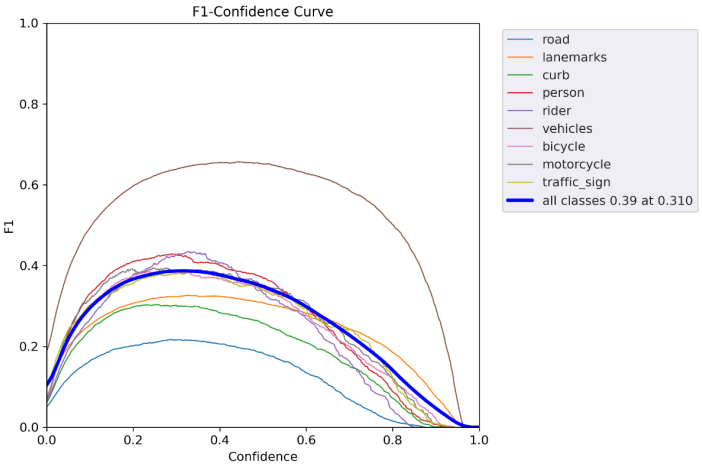
Figure 5.15 Confusion matrix after performing semantic segmentation on the rectilinear corrected dataset. The vehicles class has the highest true positive rate, while the road class has the lowest true positive rate.

Figure 5.16a illustrates the F1 curve for the bounding boxes. The F1 score for vehicles, the class that performs the best, significantly decreases after a confidence score of 0.57. The F1 score for all classes drops after a confidence score of 0.283. The F1 curve for the masks can be seen in Figure 5.16b. The F1 score for the vehicles quickly decreases after a confidence score of 0.7. For all classes, the F1 score quickly decreases after a confidence score of 0.310.

The training and validation evolution can be observed in Figure 5.17. The vali-



(a)



(b)

Figure 5.16 F1-confidence curve after performing semantic segmentation on the rectilinear corrected dataset: (a) for the bounding boxes, (b) for the segmentation masks. The vehicles class (brown) has the highest F1 score for both the bounding boxes and the segmentation masks, while the road (blue) has the worst performances.

dation box loss is around 0.083, the validation segmentation loss is approximately 0.0625, the validation object loss is 0.068, and the validation class loss is 0.02006.

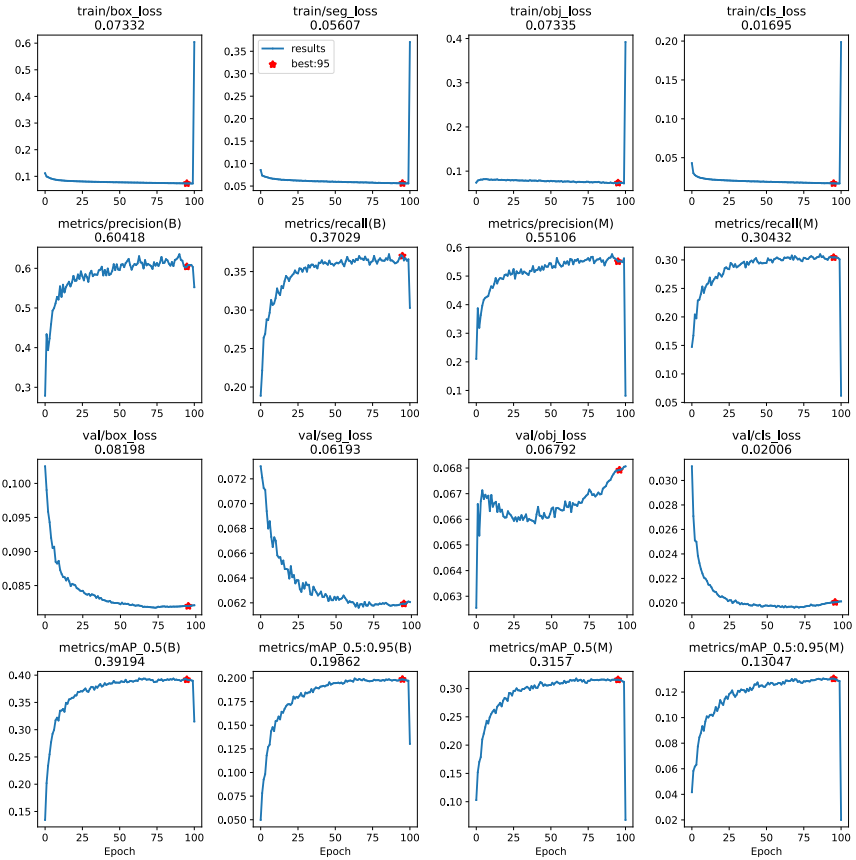


Figure 5.17 The evolution of box, segmentation, object, class loss, as well as precision, recall, mAP50, mAP50-95 for training and validation when performing semantic segmentation on the rectilinear corrected dataset

Some examples of segmentation results can be seen in Figure 5.18. Compared to Figures 5.7 and 5.12, the vehicles are better identified. On the other hand, the road is completely missed. The lanemarks are in a similar situation as well, Figure 5.18c being the only one with an identified lanemark.

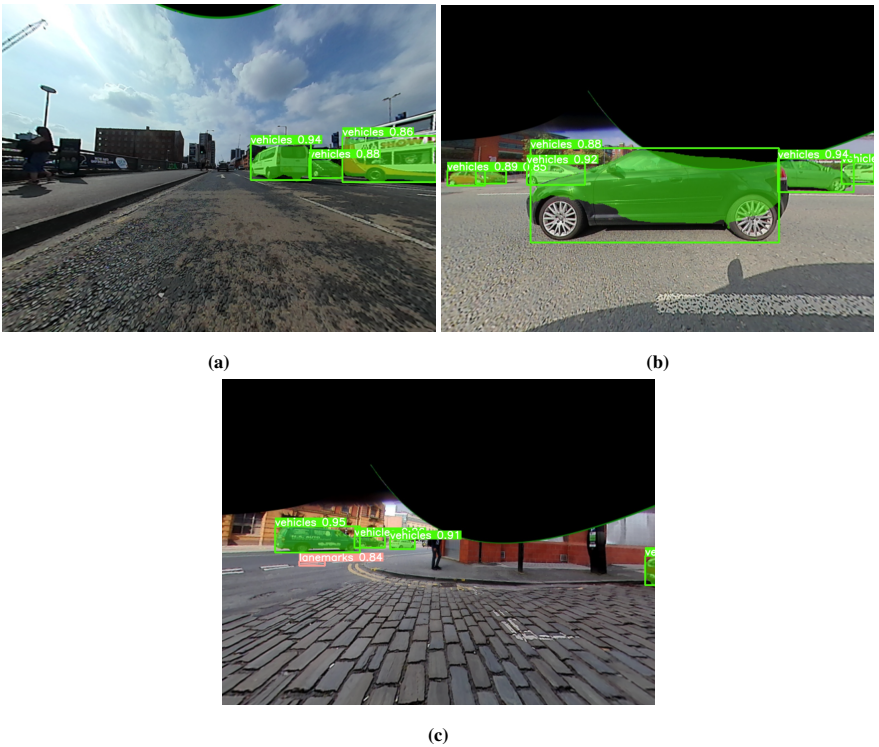


Figure 5.18 Example of segmentation result when using the rectilinear corrected images. Observe that the road and the lanmeraks are not detected. However, the bounding boxes are tighter than the bounding boxes obtained from doing object detection.

5.5 Analysis of Segmentation Results

5.5.1 Heat Map

An initial hypothesis concerning the shortcomings of our algorithm was that object detection and segmentation would be significantly worse in the corners. This came from the intuitive understanding that objects near the periphery were heavily distorted compared to objects near the center. In order to better understand the nature of the segmentation errors, and to test the hypothesis, a heatmap of the false regions across all test images was created for each individual camera view. The false regions are defined as the symmetric difference between the ground truth and the prediction masks (see Figure 5.19).

After summing the errors across all images in the test set, the log of the heatmap was calculated and subsequently rescaled to the range $[0, 255]$, as the rendering tool used expected 8-bit intensity values. The log of the image is used in an effort to increase resolution between low errors. The shifted histogram can be seen in Figure

5.20.

From Figure 5.21 it is obvious that errors are concentrated in local regions. For example, in all four images, the region where the road is most often observed has a much lower error rate than regions on the far periphery where many different and much smaller objects are commonly seen. It is not clear however that errors are concentrated at the far periphery. For example, in Figure 5.21a the highest concentration of segmentation errors seems to be concentrated at the center of the image. And while we do see a concentration of errors at the far periphery in Figures 5.21c and 5.21d, this can also be explained by the camera's downward orientation, resulting in objects far from the car being rendered primarily in these regions to begin with.

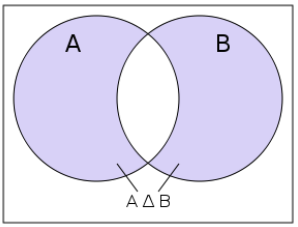
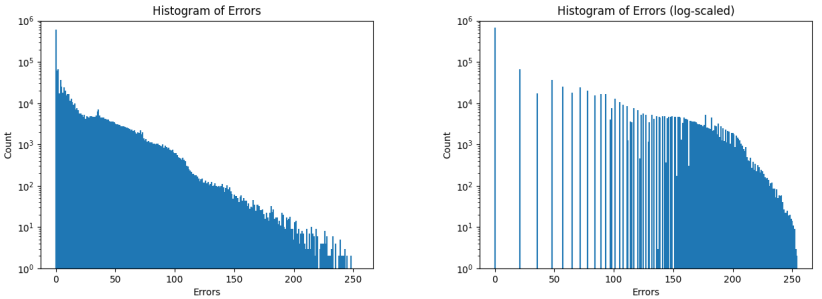
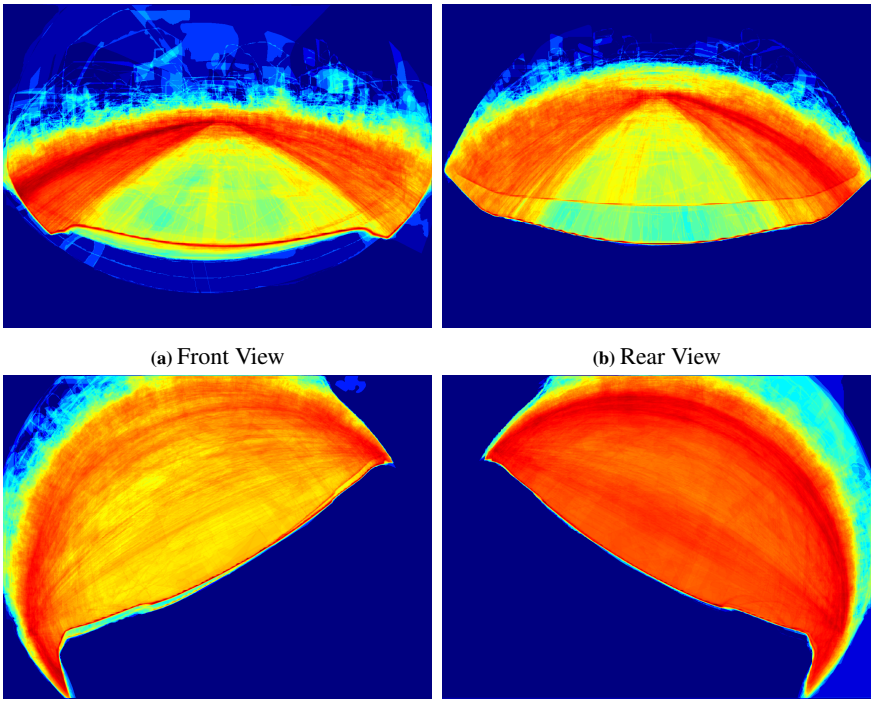


Figure 5.19 A diagram of the symmetric difference between two sets. Here A and B can be seen as the ground truth and prediction masks, respectively.



(a) Original histogram **(b)** Log-scaled histogram

Figure 5.20 Histograms of the distribution of errors before and after the logarithmic rescaling. Notice that the y-axis is shown in log-scale. With that in mind, it is apparent that the errors originally follow an exponential decay. After the log-scaling of the data, we get a distribution which spreads the low value errors across a wider range, effectively increasing the visibility of the majority of our error values.



(a) Front View

(b) Rear View

(c) Mirror View - Left

(d) Mirror View - Right

Figure 5.21 Example of segmentation result when using the original fisheye images. It can be seen in Figures 5.21c and 5.21d that errors are high for both mirror views at the periphery of the image, however, this is likely caused by the downward orientation of the camera. When the camera is orientated in this way, objects which are on the same ground plane but farther away will naturally be rendered at angles far from the principal axis of the lens.

5.5.2 Comparing the Models trained on Fisheye vs. Rectilinear Images

To better understand exactly how the two models are working, it is helpful to observe additional statistics besides the mAP. In this case, the average number of observations vs. successful detections (where the model segmented the object with an IoU w.r.t. ground truth above 0.5) was compared within the cropped field of view of the rectilinear image. An illustration is provided along with the averaged results below. Fisheye to Rectilinear (FTR) are the predictions from the fisheye model which have been mapped to the rectilinear space for comparison. Additionally, *observations* denoted in Table 5.1 are the number of times the model predicts the existence of a class, whereas *detection* denotes an IoU between prediction and ground truth above 0.5. All statistics in the table are averaged across 112 test images. From Table 5.1 it is clear that the rectilinear model is performing better

	FTR	Rect.
Num. Observed ($>$ 70% confidence)	3.14	3.62
Num. Detected (IoU $>$ 0.50)	2.29	3.13
Accuracy (mAP_{50})	0.655	0.724

Table 5.1 The average values reported were taken using the 112 test images that were found to be in both the fisheye and rectilinear image directories. Accuracies are higher than normal since 'road' and 'lane line' classes have been removed from this evaluation and predictions have been pre-filtered for those which the model gave a confidence above 70%. The main takeaway, however, should be that the rectilinear model is superior to the fisheye model within it's narrower field of view. This weakens the argument that the fisheye model is primarily failing to detect objects at the far periphery.

Figure 5.22 Areas marked in green show the false positive and false negative regions of the fisheye model's predictions. Areas marked in purple show the same for the rectilinear model, with predicted segmentations that have been mapped back to the fisheye distortion. There is no discernible difference in error between the two models that can be gathered from this image alone.

Images for the falsely segmented regions of both models were provided on the same image in Figure 5.22 and 5.23. Rectilinear false segmentations are in purple and fisheye false segmentations are in green. This explains why only green markings exist at the image's far periphery. That is, the rectilinear model, whose predictions

Figure 5.23 Supplementary to figure 5.22

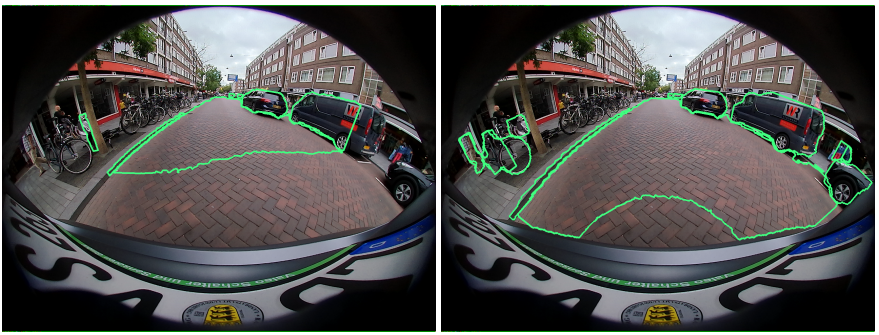
have to be mapped back to the fisheye perspective, works on a narrower field of view and thus doesn't provide predictions for areas at the far periphery.

5.6 Fair Model Comparison

When performing the above experiments with the rectified datasets, the metrics produced by YOLO do not reference the same ground truth compared to the original fisheye images. The original fisheye annotations suffer some changes due to the projection step (described earlier in Chapter 3) due to the reduction in field of view (very noticeable in Figure 5.1c). Consequently, a comparison between the above validation results will not be completely fair, since there are completely different datasets, with different annotations and field of view.

Taking into account these observations, one way to compare the performances of the models described above would be to take the predictions made by YOLO and remap them back to a fisheye image. This can be done in a similar way to the annotation conversion described in the beginning of this chapter. The world coordinates are now obtained by doing a 2D to 3D projection of the predicted cylindrical/rectilinear polygon coordinates. Afterwards, a 3D to 2D projection is done in order to get the fisheye polygon coordinates. An example of how the remapping to fisheye

looks like can be seen in Figure 5.24. The effect of the reduction in the field of view is noticeable in Figure 5.24a. Not only are some objects entirely missed (i.e. bicycles, people), but some objects also get cropped (i.e. the road, the van on the right side).



(a) (b)

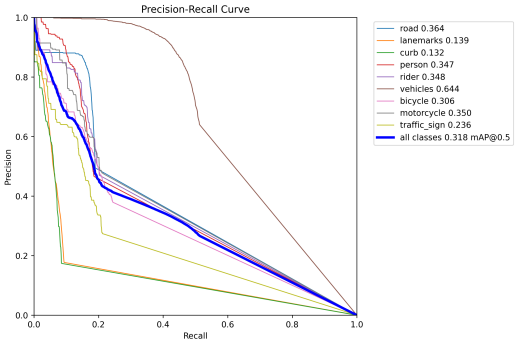
Figure 5.24 Example of segmentation result when remapping the predicted polygon coordinates to fisheye from rectilinear (a) and cylindrical (b) corrected images. Notice the effect of the reduction in the field of view in the rectilinear projection (missed or cropped objects in Figure 5.24a).

At this point, the new predictions are actually the remapped polygon coordinates and their corresponding class. The ground truth is the original fisheye ground truth. Therefore, a fair comparison can be made, since the reference is the same (the fisheye ground truth), no matter what rectification technique was used. It is also worth mentioning that the comparison uses the test set, as opposed to the validation set, which was used to produce the previous metrics.

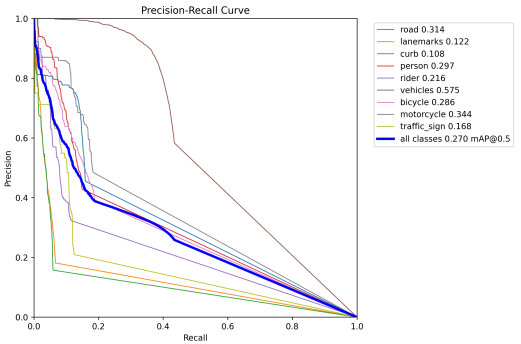
A good metric for evaluating the performance of the detection and segmentation models is mAP. We implemented a custom version of mAP in order to compare our models. Firstly, we iterated through all the predictions and computed the intersection over union (IoU) with the corresponding object in the ground truth. If the IoU score was greater than 0.5, we classified the prediction as a true positive. Then, for each class, we computed the precision and the recall, also plotting the precision-recall curves. Lastly, the mAP was computed as the averaged area under the precision-recall curve.

The results of the evaluation can be seen in Figure 5.25. Figure 5.25a presents the precision-recall curve of the test set from the model using the fisheye images. The mAP results are listed on the right side. As it can be seen, the class with highest mAP is the vehicles class, with a score of 0.644. The classes with the lowest scores are curb (0.132) and lanemarks (0.139). The mAP for all classes is 0.318. Figure 5.25b illustrates the precision-recall curve of the test set from the model using the remapped cylindrical predictions. The class with the highest mAP is the vehicles

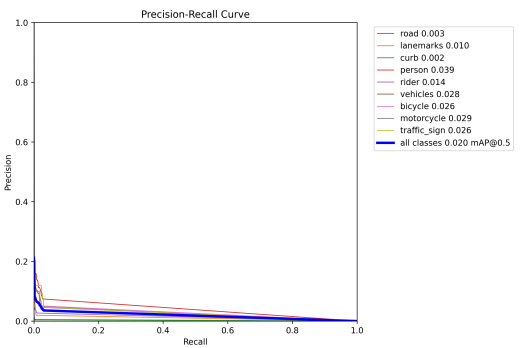
class, with a score of 0.575. The lanemarks and the curb classes have the lowest scores (0.122 and 0.108, respectively). It should be noticed that all classes have actually lower mAP scores than the mAPs of the models trained on the fisheye images. The mAP for all classes is 0.27 (vs. 0.318 with the model using the fisheye images). The precision-recall curve of the test set from the model using the remapped rectilinear images can be seen in Figure 5.25c. The class with the highest mAP is the person class (0.039), while the classes with the lowest scores are the road (0.003) and the curb (0.002). The mAP for all classes is 0.020. The poor performance is because of the reduction in the field of view.



(a)



(b)



(c)

Figure 5.25 Precision-recall curves from the test set after performing semantic segmentation: (a) - original fisheye, (b) - remapped predictions from the cylindrical images, (c) - remapped predictions from the rectilinear images. The values on the top right corner correspond to the mAP50 scores. The model trained on the fisheye images still achieves the best mAP, followed by the model trained on the cylindrical corrected images. The loss in the field of view is also noticeable in the precision-recall curve for the rectilinear predictions.

6

Tiled Rectification

This chapter describes another rectification approach, in which the fisheye image is divided into tiles. Each tile is a rectilinear mapping of the original image at varying projection angles. With this approach, the subject's scale and FOV are both maintained during rectification. YOLOv5 was again used for segmentation and training was carried out on each tile individually. In order to compare the performance of the tile approach with the above methods, it was necessary to merge the predicted segmentations from each tile in a scene.

6.1 Introduction

While the rectilinear model may have achieved the best numerical results out of the three models presented so far, the comparison is not entirely valid, as the original FOV is much smaller. Objects that were cut off by the limited FOV were also removed from the ground truth, meaning that between the fisheye model and the rectilinear model, they do not even share the same problem space. If we were to re-evaluate the rectilinear model by including the ground truth objects that were not visible due to rectification, the performance would be significantly worse, as revealed in Table 6.1.

Ground Truth	mAP_{50}
Cropped	0.315
Original	0.020

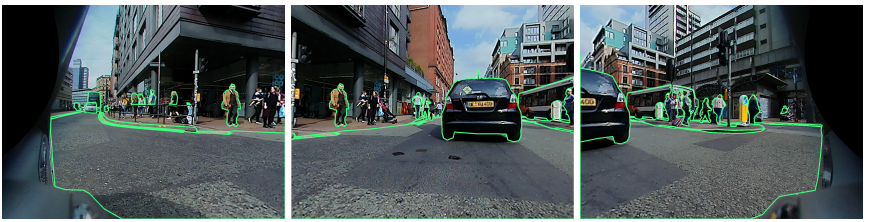
Table 6.1 The mean average precision across all images and classes of the rectilinear model. Cropped ground truth being the ground truth which only includes objects fully within the cropped FOV. Whereas, 'original' means that the model is evaluated with respect to the original ground truth.

If the field of view were maintained during rectilinear mapping, objects at the center of the image would become too small and therefore harder to detect, thus presenting new problems. Instead, a more reasonable approach might be to discretize

the problem into multiple rectilinear mappings at different angles, evenly spaced across the field of view. We will call these different mappings, 'tiles'. Considering the specific dataset we used, it was only necessary to produce three tiles, spaced horizontally, for the front and rear cameras and six tiles, composed of two horizontal rows of three tiles each, for the left and right mirror cameras.

6.2 Adapting the Annotations

In order to get the proper ground truth annotations for each tile, we use the same rectilinear mapping functions as with the image rectification, with some additional processing to handle any cropped segmentations. When mapping the 3D world coordinates to the rectilinear image space, any points with a negative Z (from the 3D world coordinates) are filtered, as this would otherwise result in inverted annotations from the opposite end of the fisheye FOV being projected as well. After getting the rectilinear image coordinates for the objects with positive Z, they are filtered again for any segmentations which lay completely outside the tile's FOV. And finally, segmentations which lay partially outside the FOV get their exterior coordinates translated to the perimeter of the image, resulting in a fully adapted ground truth.



(a) (b) (c)

Figure 6.1 Example of annotated tiles corresponding to a fisheye image. The image was taken with the front view camera.

An example of rectilinear annotated tiles can be seen in Figure 6.1. We chose not to tile the upper and lower parts of the image, as they don't have any important information (the upper part has only buildings and the lower part has only the road or parts of the car itself). Apart from this, it can be noticed that this method of rectifying the image preserves the original field of view. On the other hand, there are a lot of overlapping areas. The solution to the overlapping area is discussed with more detail in Section 6.4

The angles x, y and z, used to create the rotation matrices, are documented in Table 3.1 and include a unique set of angles depending on the camera angle. See equations 3.14, 3.15, 3.16 for reference. The sets of angles used for each camera are

$$\alpha = [-62, 0, 62], \tag{6.1}$$
$$\beta = [0, 41]. \tag{6.2}$$

6.3 YOLO Results

We generated tiles for every fisheye image, using them together with the corresponding annotations as input to YOLO. The COCO weights were used for initialization. The model was trained for 100 epochs using a batch size of 16. The training took roughly 15 hours and the validation results can be noticed in Listing 6.1. The additional time in training can be attributed to the now larger dataset composed of tiles from each original frame.

Class	Images	Instances	Box(P	R	mAP50	mAP50-95)	Mask(P	R	mAP50	mAP50-95)	
all	3669	65492	0.702	0.487	0.541	0.326	0.671	0.423	0.457	0.226	
road	3669	10766	0.784	0.485	0.576	0.43	0.715	0.402	0.44	0.317	
lanemarks	3669	26742	0.698	0.549	0.594	0.353	0.599	0.43	0.442	0.203	
curb	3669	6643	0.691	0.413	0.473	0.287	0.522	0.278	0.265	0.0889	
person	3669	3817	0.732	0.397	0.496	0.268	0.716	0.35	0.425	0.185	
rider	3669	1092	0.73	0.477	0.521	0.269	0.742	0.429	0.451	0.162	
vehicles	3669	11525	0.807	0.706	0.774	0.539	0.817	0.676	0.745	0.477	
bicycle	3669	3073	0.614	0.44	0.468	0.244	0.635	0.401	0.434	0.179	
motorcycle	3669	946	0.722	0.444	0.508	0.296	0.741	0.409	0.48	0.224	
traffic_sign	3669	888	0.54	0.47	0.461	0.252	0.548	0.43	0.428	0.196	

Listing 6.1 Semantic segmentation validation results of the rectilinear image tiles using YOLOv5 as the training model. The picture shows for each class the number of images used for validation, the number of instances, precision (P), recall (R), mAP50 and mAP50-95 scores for both the bounding boxes and the semantic segmentation.

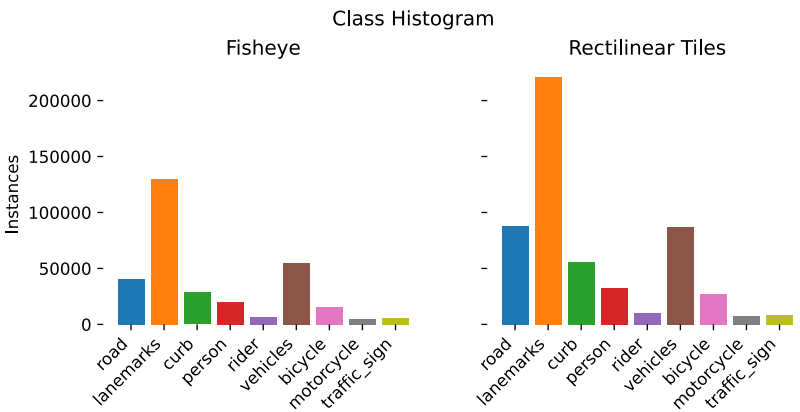
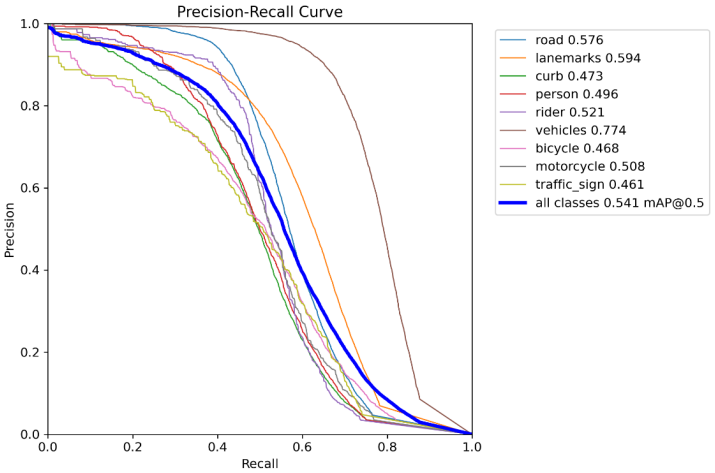
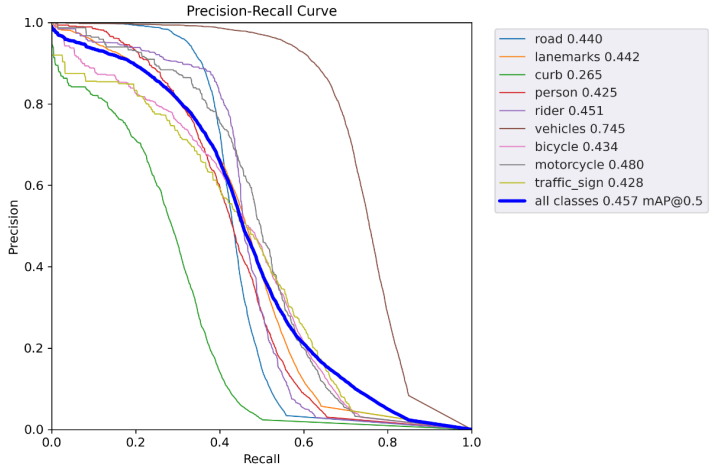


Figure 6.2 Distribution of class instances for semantic segmentation in the rectilinear image tiles (right) compared to the distribution of class instances in the original fisheye images (left). Notice the increase in the number of instances due to overlapping objects.

As it can be seen in Listing 6.1, the box precision for all classes is 0.702 and the box recall is 0.487. The mask precision for all classes is 0.671 and the mask recall is 0.423. The boxes mAP50 for all classes is 0.541, while the masks mAP50 is 0.457.



(a)



(b)

Figure 6.3 Precision-recall curve after performing semantic segmentation on the rectilinear image tiles: (a) for the bounding boxes, (b) for the segmentation masks. The results on the top right corner correspond to the mAP50 scores. The vehicles (brown) achieve the best results for both the bounding boxes and the segmentation masks. The traffic sign (light green) has the lowest AP for the bounding boxes, while the curb (green) has the lowest AP for the segmentation masks.

The number of instances in the dataset is shown in Figure 6.2. The original dataset contained 120000 instances for lanemarks and 60000 instances for vehicles. The tiles have some overlapping objects, consequently the number of lanemarks is now around 200000 and the number of vehicles is around 90000.

Figure 6.3a pictures the precision-recall curve for the bounding boxes, having the corresponding mAP50 on the right side. The highest score corresponds to the vehicles class - 0.774, while the lowest score corresponds to traffic sign class - 0.461. The precision-recall curve for the masks can be seen in Figure 6.3b. The vehicles have again the highest mAP50 - 0.745. The lowest mAP50 corresponds to the curb class - 0.265.

The confusion matrix can be seen in Figure 6.4. 72% of the vehicles are correctly classified, but only 42% of the people are correctly identified. Also, 58% of the traffic sign instances and 57% of the curb instances are missed.

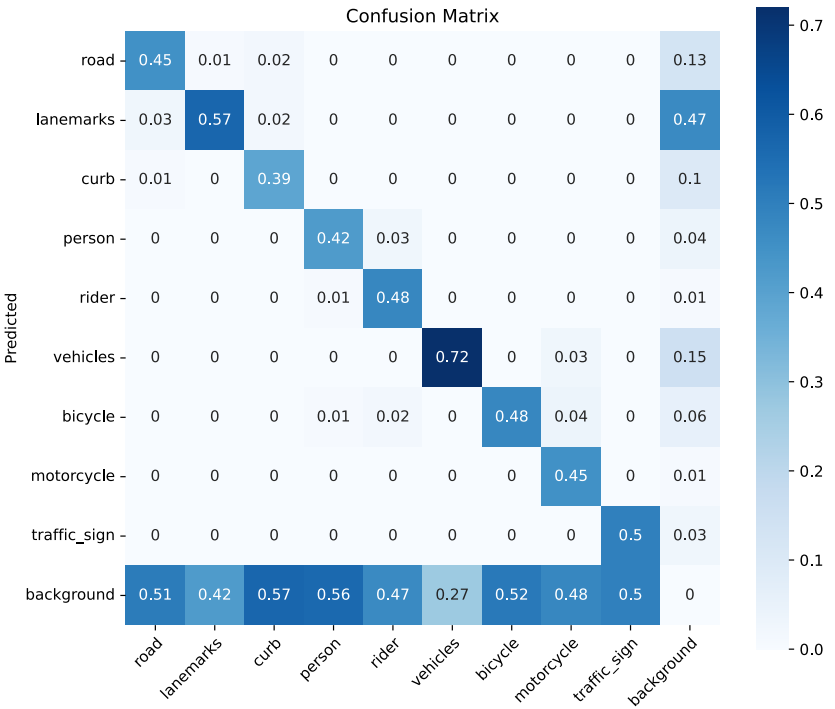
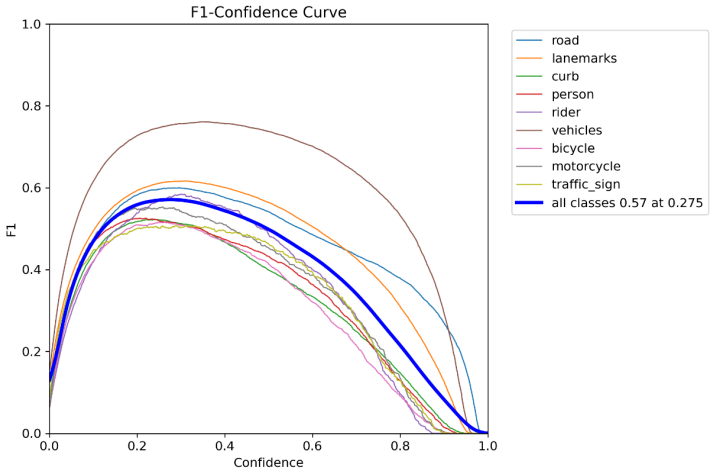
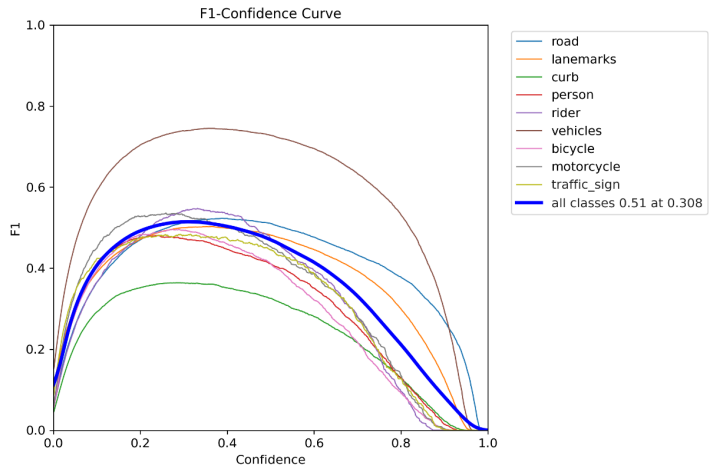


Figure 6.4 Confusion matrix after performing semantic segmentation on the rectilinear image tiles. The vehicles class has the highest true positive rate, while the curb class has the lowest rate.

Figure 6.5a illustrates the F1 curve for the bounding boxes. Once again, the class



(a)



(b)

Figure 6.5 F1-confidence curve after performing semantic segmentation on the rectilinear image tiles: (a) for the bounding boxes, (b) for the segmentation masks. The vehicles class (brown) has the highest F1 score for both the bounding boxes and the segmentation masks.

that performs the best is the vehicles class. The F1 score for the vehicles decreases after a confidence score of roughly 0.6. The F1 score for all classes drops after a confidence score of 0.275. Figure 6.5b pictures the F1 curve for the masks. The F1 score for the vehicles decreases after a confidence score of approximately 0.6, while the F1 score for all classes decreases after a confidence score of 0.308.

The training and validation evolution can be observed in Figure 6.6. The vali-

dation box loss is approximately 0.048, the validation segmentation loss is roughly 0.040, the validation object loss is around 0.059, and the validation class loss is 0.013.

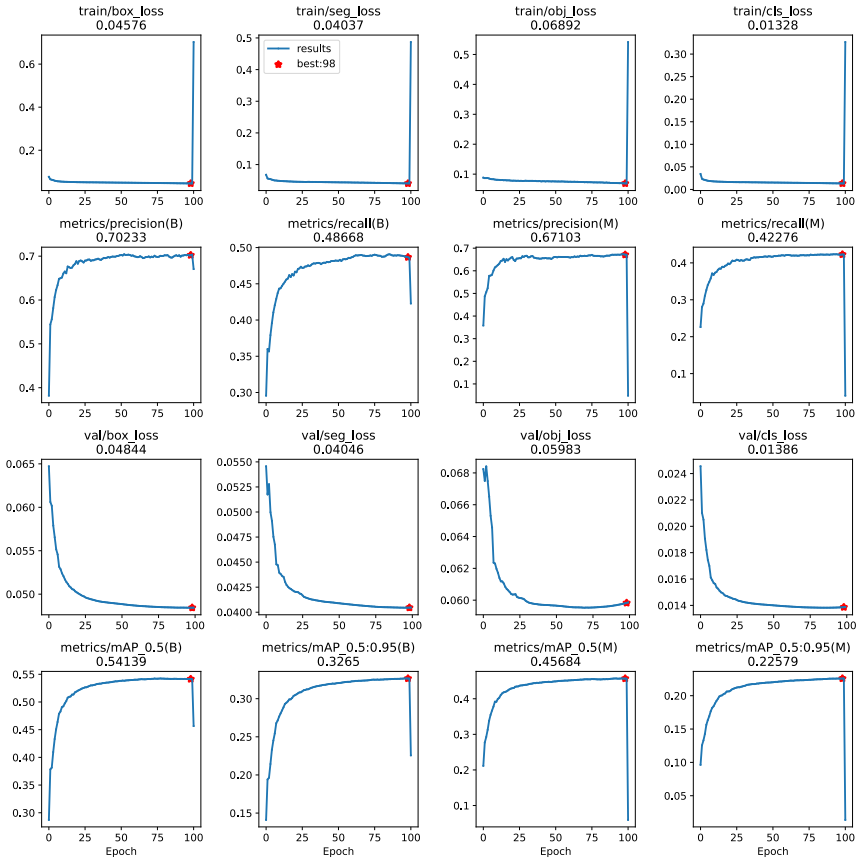


Figure 6.6 The evolution of box, segmentation, object, class loss, as well as precision, recall, mAP50, mAP50-95 for training and validation when performing semantic segmentation on the rectilinear image tiles

6.4 Tile Merging

The metrics provided above (see Section 6.3) come directly from the training process and are referencing a modified ground truth. To produce a fair comparison between the different models, we would need to use the same ground truth each time. There is also an overlap bias. That is, objects detected in the regions of tiles

which overlap with each other would be counted twice when averaging the results. Double counting instances which show up in multiple tiles could skew the results of our evaluation and therefore needs to be addressed. In order to make a fair comparison with the other models, it is necessary to merge the predicted segments from each tile into a combined collection of predictions for the entire field of view.

This is done by mapping the rectilinear tiles back to the fisheye space, taking into consideration each tile's unique rotation matrix (see Section 3.4). An example of the remapped predictions from each tile can be seen in Figure 6.8. The specific details around how overlapping segments were merged are covered in Section 6.4.1

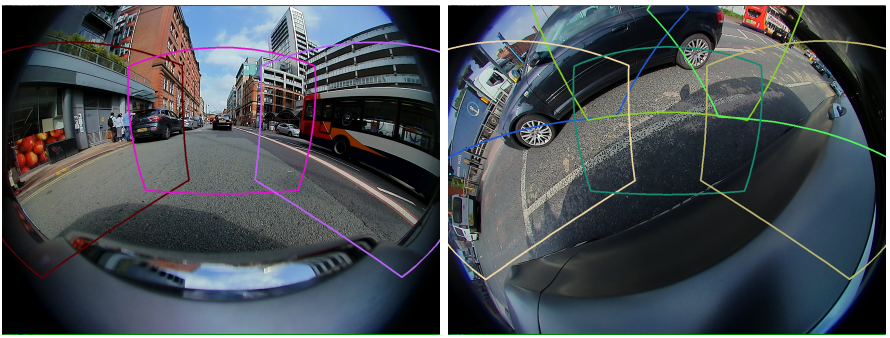
One thing to consider when mapping predictions back to the fisheye space from separate tiles is that overlapping segmentations from different tiles might predict different classes. An active approach to handling this edge case was explored, particularly for segmentations that overlapped significantly. Overlapping segmentations with an IoU greater than 0.5 and whose predicted classes are dissimilar were merged, with the highest confidence segmentation propagating their class to the merged result. However, this approach provided negligible benefit and was turned down in place of simply letting dissimilar but overlapping segmentations remain in the merged prediction space.

6.4.1 Execution

Merging the predicted segments seems a lot easier in theory than it is to carry out in practice. To best address this challenge, each segmentation, across all tiles, can be seen as a node in an unconnected graph. Our job is to connect nodes whose segmentation overlaps, at least once, with any other node's segmentation from a different tile. Thanks to the non-maximum suppression from the YOLOv5 output, there is no chance of segmentations overlapping within a single tile.

The connectedness can be stored in an adjacency matrix $\mathbf{A}_{poly} \in \mathbb{B}^{\mathbf{n}\times\mathbf{n}}$, where n is the number of segmentations across all tiles in a given scene. An additional adjacency matrix $\mathbf{A}_{tile} \in \mathbb{B}^{\mathbf{m}\times\mathbf{m}}$ is created, where m is the number of tiles, and is used to store information about which tiles are overlapping. A_{tile} remains constant respective to the given camera angle. Since the front and rear camera's use only three tiles and the side mirror cameras use six, different adjacency matrices were used respectively. Representations of the tile overlap can be seen in Figure 6.7 and the associated adjacency matrices A_{poly} can be found under Equation 6.3:

$$A_{poly} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \qquad A_{poly} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}. \tag{6.3}$$



(a) Front View (b) Mirror View – Right

Figure 6.7 Representation of the area that each rectilinear tile covers in the fisheye space. The front and rear cameras only have three tiles, as can be seen in Figure 6.7a. The two side mirror cameras use six tiles due to their unique orientation w.r.t. the road, as can be seen in Figure 6.7b.

(a) Unmerged segmentations (b) Merged segmentations

Figure 6.8 An illustration of the merging of segments from the separate tiles into one homogenous set of segmentations. The left image includes unmerged segmentations from all tiles, mapped back to the fisheye space, and the right image shows the resultant set of segmentations after merging any overlapping instances. Due to our specific tiling arrangement, there is clearly a missing segmentation for part of the road just in front of or behind the car. This could easily be addressed with another row of three tiles, as with the side mirror cameras. Note that coloring is only meant to distinguish between segmentation instances and is not representative of the instance's class.

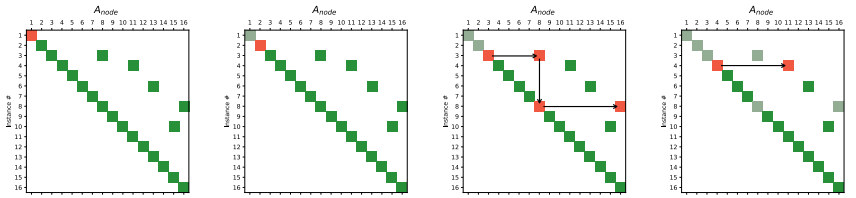


Figure 6.9 Illustration of the depth first search algorithm used to merge overlapping segmentations from the different tiles. The algorithm iterates through the adjacency matrix A_{poly} starting at the first segmentation instance and performs a depth-first search for any instances which overlap it (seen in Figure 6.9c). Shown above are the first four iterations of the algorithm, going through each instance sequentially. Any previously observed instance is saved in memory (as shown in gray) and skipped at future iterations of the loop.

The last step is to run through the newly constructed adjacency matrix A_{poly} for the given scene and merge the overlapping segmentations. The algorithm which inspired our approach is called depth-first search, but unlike our problem, pertains to graphs which are fully connected [47]. For our particular problem, the depth-*wise* search is best illustrated in Figure 6.9. In short, the matrix iterates through each segmentation and searches for other connected segmentations depth-wise, merges those segmentations if necessary and finally stores the merged segmentations in memory so that they may be skipped at a later stage.

Once the merging is complete, we use the same process as with all the other models to compute the average precision (AP) and mean-average precision (mAP) performance metrics. The theory for which is covered in Section 5.6. Results from predictions on the test set showed a dramatic improvement over any previously tested method. A plot of the precision-recall curve can be seen on Figure 6.10.

From these results, it is again made clear that convolutional neural networks perform best on rectilinear images.

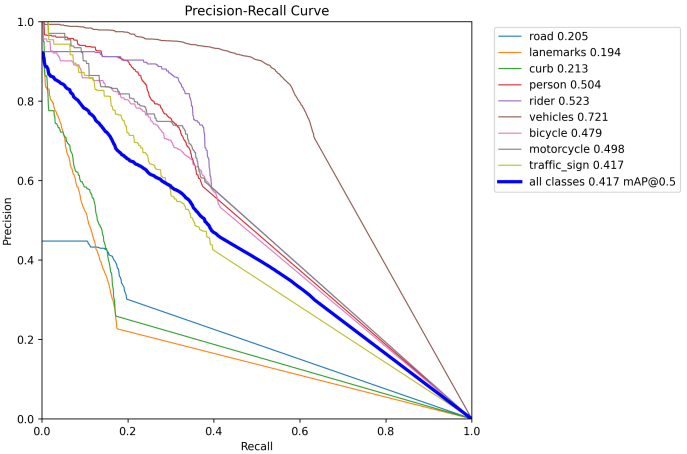


Figure 6.10 Plot of the precision-recall curve for the tiled rectification method. The numbers in the top right corner correspond to the mAP scores. This method performed best in all classes except for **road**. This is likely due to the tile configuration used for the front and rear cameras as seen in Figure 6.7a, which excludes a section of road nearest to the vehicle.

7

Conclusion and Discussion

7.1 Summary of Experiments

In this thesis, we aimed to develop an improved way of doing object detection and segmentation with fisheye distorted images. We chose to use the WoodScape dataset, first analyzing the ground truth for the detection and segmentation tasks and discovering some labeling issue.

7.1.1 Object Detection

In the first stage of our project, object detection models were compared. The original WoodScape dataset was used as a baseline to test against two different rectification models; cylindrical and rectilinear. On the one hand, cylindrical rectification of the image seems to be a suitable approach because it preserves most of the initial fisheye field of view while eliminating vertical distortions. On the other hand, the results obtained after running YOLO with the cylindrical images are similar to the results obtained with the initial fisheye images. Consequently, there was no improvement from the object detection point of view.

Model	mAP50
Fisheye	0.49
Cylindrical	0.47
Rectilinear	**0.54**

Table 7.1 The mAP50 scores for the object detection models trained on fisheye, cylindrical and rectilinear images. The best results are achieved by the model using the rectilinear corrected images. However, this is not a fair comparison, since the ground truth is not the same for all models due to the change of FOV in the rectified images.

Just by looking at the numerical results from Table 7.1, one could say that the object detection results are better with the rectilinear corrected image, but they cut a lot of information from the pictures. Therefore, the ground truth is not the same as the original one.

Another aspect that is worth noticing is that the bounding boxes for the cylindrical and the rectilinear corrected images are too large, covering a lot of background as well (see Figures 4.16 and 4.22). This issue stems from the effect undistortion has on the original annotations from the fisheye images. One way to solve this issue would be to actually perform semantic segmentation instead of object detection and extract the bounding box coordinates from the segmentation mask.

7.1.2 Semantic Segmentation

For this next stage of the project, semantic segmentation models were trained using a completely different ground truth. The shift, originating from a desire to have better annotations after rectifying the images.

Our baseline for segmentation had the fisheye images and the generated annotations for semantic segmentation. We generated the semantic segmentation annotations for the cylindrical and the rectilinear versions. From the numerical results, it looked like the model trained on the rectilinear images performed the best. We did some in depth analysis of the missed objects and the accuracy of the predictions in order to better understand the results. However, by performing the projections to the cylindrical and rectilinear domains, we ended up with different ground truths for the three models. Consequently, we remapped the predictions from the test sets back to the fisheye space in order to be able to fairly compare the models. We implemented a custom version of mAP for the remapped predictions. When analyzing the results of the remapped predictions, the model using rectilinear images did not achieve reasonable results due to the drastic reduction in the FOV.

Therefore, we needed to implement a different rectification method that was able to both give rectilinear images and to maintain the original FOV. In order to achieve these goals, we divided the original image into 3 or 6 tiles (depending on the camera orientation) and we perform a rectilinear rectification for each one of them. We generated the annotations for the tiles, and we fed the new data into YOLOv5 model. Then we merged the tiles from the test set, and we remapped the predictions back to fisheye, making sure that there are no overlapping predictions.

The combined results of our four models for semantic segmentation can be observed in Figure 7.1. These results come from evaluating the remapped predictions from the test sets. The graph pictures the precision-recall curve and the mAP scores can be seen on the right side. The best score is achieved by the model using the tiles, having an mAP50 of 0.417. The second-best model is the one trained on the fisheye images (i.e., our baseline), with a mAP50 of 0.318. Even though the cylindrical correction keeps the original FOV, the performance of the model is actually worse than the baseline, having a mAP50 of 0.270. The model trained on the rectilinear images has the lowest mAP50, 0.020. The poor result is due to the reduction in the FOV.

Another aspect worth mentioning is that when we did the tile rectification, we actually prioritized all the other classes, except for the road (as we actually decided

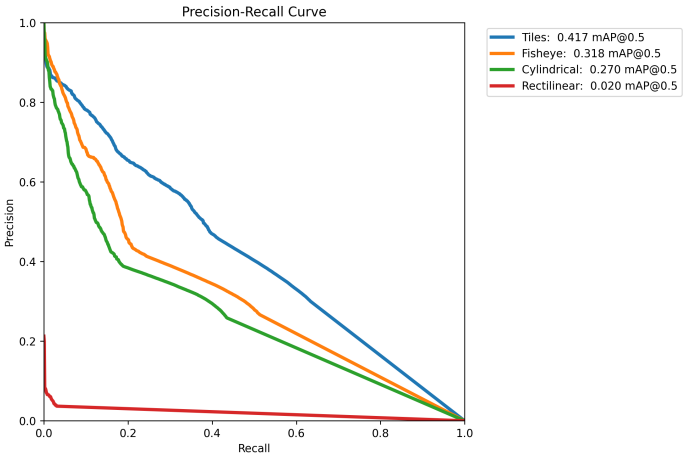


Figure 7.1 Precision-recall curve for the four developed models using the tiles (blue line), original fisheye (orange line), cylindrical corrected (green line), and rectilinear corrected (red line) images. The mAP scores are shown on the right side of the picture. The best results are achieved by the model trained on the tiles (blue). The models trained on the cylindrical (green) and rectilinear (red) corrected images achieve lower mAPs than the model trained on fisheye images (orange).

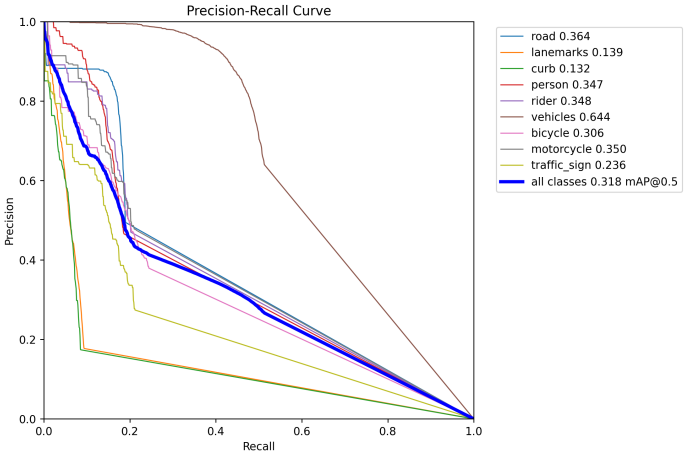
to cut parts of it when dividing the picture). An example of this can be seen in Figure 6.8b.

Figure 7.2 shows the precision-recall curves, as well as the AP results for every class. As it can be seen in Figure 7.2d, all the classes except for the road have higher APs than the baseline (Figure 7.2a).

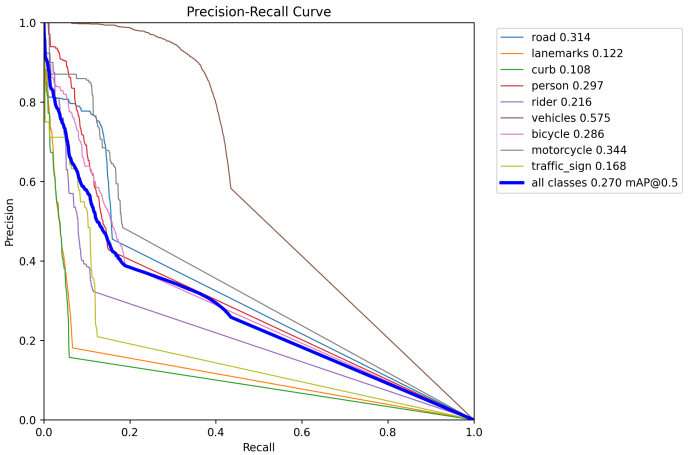
By extracting the AP for the road class from all the models, the new mAP50 can be seen in Table 7.2. As it can be seen, the model trained on the tiles has the best score, with a mAP50 of 0.44. The second-best model is the baseline, with a mAP50 of 0.31.

Model	mAP50
Fisheye	0.31
Cylindrical	0.26
Rectilinear	0.02
Tiles	**0.44**

Table 7.2 The mAP50 scores for the segmentation models trained on fisheye, cylindrical, rectilinear, tiles images when not taking into consideration the AP for the road class. The best results are again achieved by the model using the tiles.

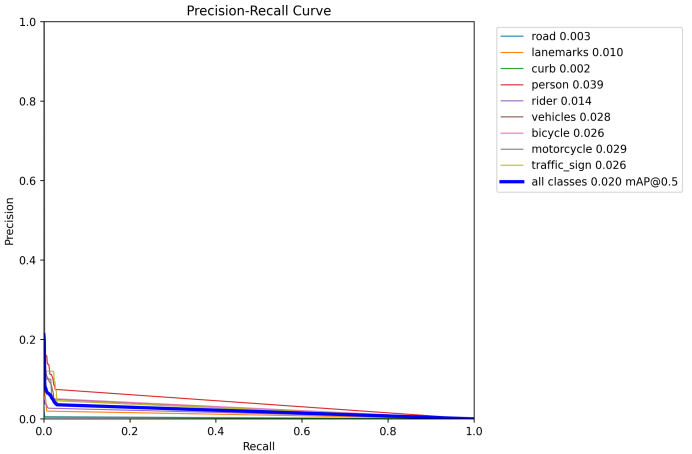


(a)

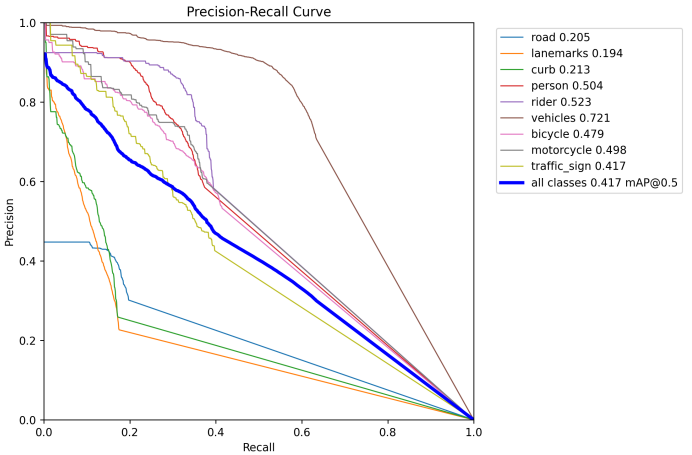


(b)

Figure 7.2 Precision-recall curve for the four developed models: (a) model trained on fisheye images, (b) model trained on cylindrical corrected images, (c) model trained on rectilinear corrected images, (d) model trained on the tiles. The results in the top right corner correspond to the mAP scores. The model trained on the tiles achieved the best AP for all classes, except for the road. The models trained on the cylindrical and rectilinear corrected images achieve lower APs for each class compared to the model trained on the fisheye images.



(c)



(d)

Figure 7.2 Precision-recall curve for the four developed models: (a) model trained on fisheye images, (b) model trained on cylindrical corrected images, (c) model trained on rectilinear corrected images, (d) model trained on the tiles. The results in the top right corner correspond to the mAP scores. The model trained on the tiles achieved the best AP for all classes, except for the road. The models trained on the cylindrical and rectilinear corrected images achieve lower APs for each class compared to the model trained on the fisheye images.

7.1.3 Bounding Box Comparison

While we noticed that the boxes obtained from the segmentation masks are tighter, it would also be interesting to compare the mAP50 for the detection and segmentation w.r.t the bounding boxes.

There are four common classes for both segmentation and detection: vehicles, person, bicycle, traffic_sign. In order to compare the results, we take the APs for these classes and we average them in order to get the mAP50 score. We follow this procedure for the three models that we presented before.

Model	mAP50 detection	mAP50 segmentation
Fisheye	0.508	0.397
Cylindrical	0.500	0.351
Rectilinear	0.552	0.442

Table 7.3 The bounding box mAP50 scores for the models trained on fisheye, cylindrical, rectilinear images when taking into consideration the APs for the vehicles, person, bicycle, traffic_sign classes. The results for the bounding boxes generated from the segmentation masks are lower. Notice that this is the case also for the model using the fisheye image, which made us suspect that there are problems with the way the images have been annotated.

As it can be noticed in Table 7.3, the mAP50 of the bounding boxes from detection are higher than the ones resulted from the segmentation masks. What is even more interesting, is that this behavior is the same also when using fisheye distorted images. This can indicate that there is a divergence between the way the pictures have been annotated for the detection task vs. the segmentation task.

7.2 Future Work

While we may have been successful in improving the semantic segmentation of fisheye data with our tiling method, other methods which do not require multiple rectifications might be less computationally expensive. Therefore, our next line of research would be in training a more advanced neural network architecture which does not exhibit the same translational invariance constraints as a CNN.

One such architecture which we have identified is deformable convolution, as mentioned in Section 2.2. Such a method would potentially reduce the computational overhead of rectifying multiple tiles from each frame. A similar architecture called FisheyeHDK, mentioned in the same section, uses the deformable convolution method but constrains the problem slightly by first embedding the learned offsets into a hyperbolic Poincare ball. After corresponding with one of the researchers on the project, we are more confident in implementing our own version of the same FisheyeHDK architecture. Successful implementation will require custom gradient functions for updating the learned offsets in PyTorch.

Lastly, it is hard not to mention the recent successes of transformer networks. By tokenizing all forms of data from language, LiDAR and image data, the transformer network architecture has proven itself to be effective in a plurality of domains (see [48] as a strong example of this). It has even proven to be an applicable architecture for our particular task when a transformer based network, called the Swin Transformer [49], won a semantic segmentation challenge using the very same WoodScape dataset [50]. With further research into the computational efficiency of the methods we have previously explored, it would be interesting to compare these to both transformers and deformable convolution methods.

Bibliography

[1] S. K. Yogamani, C. Hughes, J. Horgan, *et al.*, "WoodScape: A multi-task, multi-camera fisheye dataset for autonomous driving," *CoRR*, vol. abs/1905.01489, 2019. arXiv: 1905.01489. [Online]. Available: http://arxiv.org/abs/1905.01489.

[2] R. Wood, "XXIII. Fisheye views, and vision under water," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 12, no. 68, pp. 159–162, 1906. DOI: 10.1080/14786440609463529. eprint: https://doi.org/10.1080/14786440609463529. [Online]. Available: https://doi.org/10.1080/14786440609463529.

[3] P. Hu, J. Yang, L. Guo, X. Yu, and W. Li, "Solar-tracking methodology based on refraction-polarization in Snell's window for underwater navigation," *Chinese Journal of Aeronautics*, vol. 35, no. 3, pp. 380–389, 2022, ISSN: 1000-9361. DOI: https://doi.org/10.1016/j.cja.2021.02.011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1000936121000704.

[4] Wikipedia contributors, *Snell's window — Wikipedia, the free encyclopedia*, [Online; accessed 27-March-2023], 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Snell%27s_window&oldid=1060137509.

[5] X. Ying, Z. Hu, and H. Zha, "Fisheye Lenses Calibration Using Straight-Line Spherical Perspective Projection Constraint," in *Computer Vision – ACCV 2006*, P. J. Narayanan, S. K. Nayar, and H.-Y. Shum, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 61–70, ISBN: 978-3-540-32432-4.

[6] X. Shouzhang and W. Fengwen, "Generation of Panoramic View from 360 Degree Fisheye Images Based on Angular Fisheye Projection," Oct. 2011. DOI: 10.1109/DCABES.2011.49.

[7] Y. Xiong and K. Turkowski, "Creating image-based VR using a self-calibrating fisheye lens," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1997, pp. 237–243. DOI: 10.1109/CVPR.1997.609326.

[8] S. Urban, J. Leitloff, S. Wursthorn, and S. Hinz, "Self-Localization of a Multi-Fisheye Camera Based Augmented Reality System in Textureless 3d Building Models," *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 1, pp. 43–48, Oct. 2013. DOI: 10.5194/isprsannals-II-3-W2-43-2013.

[9] A. R. Sekkat, Y. Dupuis, V. R. Kumar, *et al.*, "SynWoodScape: Synthetic Surround-View Fisheye Camera Dataset for Autonomous Driving," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 8502–8509, Jul. 2022. DOI: 10.1109/lra.2022.3188106. [Online]. Available: https://doi.org/10.1109%2Flra.2022.3188106.

[10] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *CoRR*, vol. abs/1506.02640, 2015. arXiv: 1506.02640. [Online]. Available: http://arxiv.org/abs/1506.02640.

[11] Z. Zhao, P. Zheng, S. Xu, and X. Wu, "Object Detection with Deep Learning: A Review," *CoRR*, vol. abs/1807.05511, 2018. arXiv: 1807.05511. [Online]. Available: http://arxiv.org/abs/1807.05511.

[12] Z. Zou, Z. Shi, Y. Guo, and J. Ye, "Object Detection in 20 Years: A Survey," *CoRR*, vol. abs/1905.05055, 2019. arXiv: 1905.05055. [Online]. Available: http://arxiv.org/abs/1905.05055.

[13] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, "A Review of Yolo Algorithm Developments," *Procedia Computer Science*, vol. 199, pp. 1066–1073, 2022, The 8th International Conference on Information Technology and Quantitative Management (ITQM 2020 2021): Developing Global Digital Economy after COVID-19, ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2022.01.135. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050922001363.

[14] F. Sultana, A. Sufian, and P. Dutta, "A Review of Object Detection Models based on Convolutional Neural Network," *CoRR*, vol. abs/1905.01614, 2019. arXiv: 1905.01614. [Online]. Available: http://arxiv.org/abs/1905.01614.

[15] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013. arXiv: 1311.2524. [Online]. Available: http://arxiv.org/abs/1311.2524.

[16] C. Szegedy, W. Liu, Y. Jia, *et al.*, "Going Deeper with Convolutions," *CoRR*, vol. abs/1409.4842, 2014. arXiv: 1409.4842. [Online]. Available: http://arxiv.org/abs/1409.4842.

[17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[18] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," *CoRR*, vol. abs/1612.08242, 2016. arXiv: 1612.08242. [Online]. Available: http://arxiv.org/abs/1612.08242.

[19] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *CoRR*, vol. abs/1804.02767, 2018. arXiv: 1804.02767. [Online]. Available: http://arxiv.org/abs/1804.02767.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[21] G. Jocher, *YOLOv5 by Ultralytics*, version 7.0, May 2020. DOI: 10.5281/zenodo.3908559. [Online]. Available: https://github.com/ultralytics/yolov5.

[22] H. Rashed, E. Mohamed, G. Sistu, *et al.*, "Generalized Object Detection on Fisheye Cameras for Autonomous Driving: Dataset, Representations and Baseline," *CoRR*, vol. abs/2012.02124, 2020. arXiv: 2012.02124. [Online]. Available: https://arxiv.org/abs/2012.02124.

[23] J. H. Hosang, R. Benenson, and B. Schiele, "Learning non-maximum suppression," *CoRR*, vol. abs/1705.02950, 2017. arXiv: 1705.02950. [Online]. Available: http://arxiv.org/abs/1705.02950.

[24] Y.-C. Su and K. Grauman, "Kernel transformer networks for compact spherical convolution," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019.

[25] O. Ahmad and F. Lecue, *FisheyeHDK: Hyperbolic Deformable Kernel Learning for Ultra-Wide Field-of-View Image Recognition*, 2022. DOI: 10.48550/ARXIV.2203.07255. [Online]. Available: https://arxiv.org/abs/2203.07255.

[26] J. W. Anderson, "Other Models of the Hyperbolic Plane," in *Hyperbolic Geometry*. London: Springer London, 1999, pp. 95–110, ISBN: 978-1-4471-3987-4. DOI: 10.1007/978-1-4471-3987-4_4. [Online]. Available: https://doi.org/10.1007/978-1-4471-3987-4_4.

[27] J. W. Anderson, "The General Möbius Group," in *Hyperbolic Geometry*. London: Springer London, 1999, pp. 19–56, ISBN: 978-1-4471-3987-4. DOI: 10.1007/978-1-4471-3987-4_2. [Online]. Available: https://doi.org/10.1007/978-1-4471-3987-4_2.

[28] L. Deng, M. Yang, H. Li, T. Li, B. Hu, and C. Wang, "Restricted Deformable Convolution based Road Scene Semantic Segmentation Using Surround View Cameras," 2018. DOI: 10.48550/ARXIV.1801.00708. [Online]. Available: https://arxiv.org/abs/1801.00708.

[29] H. Gao, X. Zhu, S. Lin, and J. Dai, *Deformable Kernels: Adapting Effective Receptive Fields for Object Deformation*, 2019. DOI: 10.48550/ARXIV.1910.02940. [Online]. Available: https://arxiv.org/abs/1910.02940.

[30] Z. Duan, O. Tezcan, H. Nakamura, P. Ishwar, and J. Konrad, "RAPiD: Rotation-Aware People Detection in Overhead Fisheye Images," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 2020.

[31] C. Huynh, A. T. Tran, K. Luu, and M. Hoai, "Progressive Semantic Segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2021, pp. 16 755–16 764.

[32] L. Li, T. Zhou, W. Wang, J. Li, and Y. Yang, "Deep Hierarchical Semantic Segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2022, pp. 1246–1257.

[33] Á. Sáez, L. M. Bergasa, E. López-Guillén, *et al.*, "Real-Time Semantic Segmentation for Fisheye Urban Driving Images Based on ERFNet," *Sensors*, vol. 19, no. 3, 2019, ISSN: 1424-8220. DOI: 10.3390/s19030503. [Online]. Available: https://www.mdpi.com/1424-8220/19/3/503.

[34] E. Romera, J. M. Álvarez, L. M. Bergasa, and R. Arroyo, "ERFNet: Efficient Residual Factorized ConvNet for Real-Time Semantic Segmentation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 1, pp. 263–272, 2018. DOI: 10.1109/TITS.2017.2750080.

[35] M. Cordts, M. Omran, S. Ramos, *et al.*, "The Cityscapes Dataset for Semantic Urban Scene Understanding," *CoRR*, vol. abs/1604.01685, 2016. arXiv: 1604.01685. [Online]. Available: http://arxiv.org/abs/1604.01685.

[36] Amazon. "Amazon Elastic Compute Cloud: User Guide for Linux Instances." (), [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html. (accessed: 13.12.2022).

[37] Amazon. "Amazon Machine Images AMI." (), [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html. (accessed: 13.12.2022).

[38] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized Intersection over Union," Jun. 2019.

[39] P. Henderson and V. Ferrari, "End-to-End Training of Object Class Detectors for Mean Average Precision," Mar. 2017, pp. 198–213, ISBN: 978-3-319-54192-1. DOI: 10.1007/978-3-319-54193-8_13.

[40] Y. Sasaki, "The truth of the F-measure," *Teach Tutor Mater*, Jan. 2007.

[41] V. R. Kumar, *WoodScape*, https://github.com/valeoai/WoodScape, 2019.

[42] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2004. DOI: 10.1017/CBO9780511811685.

[43] P.-E. Sarlin, M. Dusmanu, J. L. Schönberger, *et al.*, "LaMAR: Benchmarking Localization and Mapping for Augmented Reality," in *ECCV*, 2022.

[44] R. Padilla, S. L. Netto, and E. A. B. da Silva, "A Survey on Performance Metrics for Object-Detection Algorithms," in *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, 2020, pp. 237–242. DOI: 10.1109/IWSSIP48289.2020.9145130.

[45] K. Oksuz, B. C. Cam, E. Akbas, and S. Kalkan, "Localization Recall Precision (LRP): A New Performance Metric for Object Detection," in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., Cham: Springer International Publishing, 2018, pp. 521–537, ISBN: 978-3-030-01234-2.

[46] D. Powers and Ailab, "Evaluation: From precision, recall and F-measure to ROC, informedness, markedness & correlation," *J. Mach. Learn. Technol*, vol. 2, pp. 2229–3981, Jan. 2011. DOI: 10.9735/2229-3981.

[47] Shimon Even, *Graph Algorithms*, 2nd ed., Guy Even, Ed. Cambridge, England: Cambridge University Press, Sep. 2011.

[48] D. Driess, F. Xia, M. S. M. Sajjadi, *et al.*, *PaLM-E: An Embodied Multimodal Language Model*, 2023. arXiv: 2303.03378 [cs.LG].

[49] Z. Liu, Y. Lin, Y. Cao, *et al.*, *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*, 2021. arXiv: 2103.14030 [cs.CV].

[50] S. Ramachandran, G. Sistu, J. McDonald, and S. Yogamani, *WoodScape Fisheye Semantic Segmentation for Autonomous Driving – CVPR 2021 OmniCV Workshop Challenge*, 2021. arXiv: 2107.08246 [cs.CV].