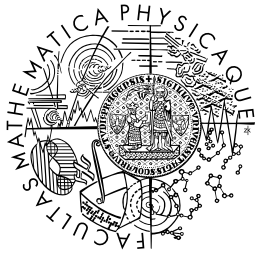


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Roman Staněk

System for automatic size and type control of cars rims

Department of Software and Computer Science Education

Supervisor of the master thesis: Ing. Adam Novozámský, Ph.D.
Study programme: Computer Science
Study branch: Artificial Intelligence

Prague 2022

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

First and foremost, I would like to express my gratitude to my supervisor Adam Novozámský, who came up with this topic, collected the video data and guided me through the wonderful field that computer vision is.

My special thanks belong to my parents for supporting me all the way through my studies, my grandparents and siblings for helping me whenever I needed it, my amazing girlfriend for her support and reviews of this work, and my friend Jan Palášek for providing his view on various computer vision topics.

Also, I would like to thank my other friends, who always patiently listened to my complaints during my studies and writing of the thesis.

In the end, I would like to thank all the people, who made their contribution to science or technology, pushing humanity towards a better future.

Title: System for automatic size and type control of cars rims

Author: Roman Staněk

Department: Department of Software and Computer Science Education

Supervisor: Ing. Adam Novozámský, Ph.D., The Czech Academy of Sciences, Institute of Information Theory and Automation, Prague, Czechia

Abstract: At the end of every automotive assembly line, there is a quality control process where factory workers check produced cars for potential defects. The computer vision field, especially neural networks for images, have great potential to complement human staff in order to produce as safe and reliable cars as possible. In this thesis we focus on the validation, whether all four wheels on a single car match in size and type. We introduce and experiment with both neural networks and traditional computer vision techniques. The approach we use is to first detect the car then classify its wheels and try to estimate their size. In the end we build a functional prototype of the system that is running in real-time. The data for this thesis were recorded in Škoda Auto factory in Mladá Boleslav in cooperation with the company.

Keywords: automation, image processing, classification, deep learning, tracking

Contents

Introduction

In 2020, approximately 480 000 cars were produced in the Škoda Auto plant in Mladá Boleslav [53]. Every car passes through the quality control process at the end of the assembly line, which ensures that the car will not leave the manufacture with serious defects that could potentially endanger customers or significantly reduce the lifespan of the car.

Most of the quality check tasks are performed by trained workers, who may be subject to fatigue, or other factors impacting the reliability of the quality check. This provides a great opportunity for automation via computer vision, which has the potential to lower the cost of the overall process and at the same time achieve super-human accuracy.

This thesis is focused on the problem of verifying the correctness of the mounting of the rims. Rims come in different sizes and shapes. On a properly assembled car, the rims on all four wheels should be the same. However, during the assembly, there is a possibility that a wrong rim could be attached to the car, which can harm the stability of the car on the road and impact the safety of the customer.

Therefore we aim to develop a proof of concept of a real-time system with the ability to automatically detect whether all four rims on an assembled car match, meaning they are of the same type and the same size.

Thesis Structure

The first chapter introduces the domain of car rims, describes the collection of the data and shows challenging properties of the data.

In the second chapter we discuss approaches that can be used in car and rims detection, build a dataset and experiment with two object detection models.

The third chapter is devoted to classification of rims based on their shape and color. We build datasets and then test prediction accuracy of multiple models.

The fourth chapter presents our knowledge of size of a rim in connection with other car components and proposes a method that can be provide a rough estimation.

The fifth chapter combines results of previous chapters to build a proof of concept of a system that is able to track cars and decide whether all rims on a car match.

The last chapter evaluates the outcomes of previous chapters and introduces ideas and possible extensions that can be employed in the future work.

1. Problem Analysis

1.1 Rims and their Visual Appearance

First, we provide an exact definition for a **wheel**, a **rim** and a **tire**, because in common language these terms can be ambiguous. In this thesis, we will call a wheel a combination of a rim and a tire. A rim is the rigid core of a wheel usually made from metal. A tire is mounted on the rim and ensures good contact with the surface under the car. It is usually made of rubber-like material. For visualization see Figure 1.1.

The main purpose of a rim is to provide rigid support for a tire and to transmit forces that influence the movement of the car, for example transmit rotational force coming from the engine to the tire [27].

Figure 1.1: Visualization of a rim and a tire.

1.1.1 Rim Structure and Mounting

The important structural components of the rim considered in this work are depicted in Figure 1.2. In the middle of the rim, there is a **center hole**, surrounded by the **center disc**. From the center disc we see **spokes** going to the outer circumference of the rim. On the circumference there is the **valve stem** which is used for inflating the tire. In the center disc, there are **bolt holes**.

The mounting of the wheel follows a specific procedure. First, the wheel is put on the wheel hub and centered using the center hole, afterwards it is secured with wheel bolts.

The wheel bolts form a pattern that can be defined by two numbers. The first describes the number of bolts. The second is the pitch circle diameter(PCD), which denotes the diameter of a circle going through the centers of the bolts, see

Figure 1.2: Rim structure - center hole is marked yellow, center disc is green, spokes are blue, valve stem is orange and bolt holes are red.

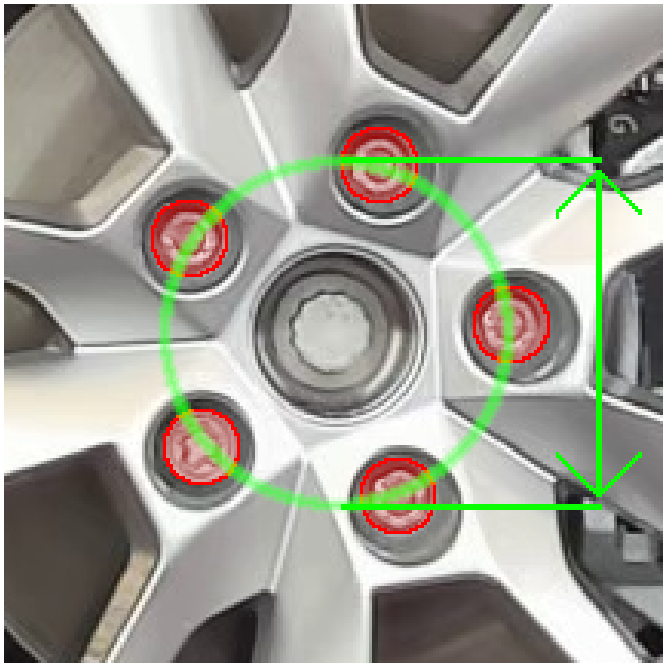


Figure 1.3: PCD vizualization - pitch circle is the green circle that passes through the centers of the wheel bolts (which are marked red).

Figure 1.3. It is measured in millimeters. For example, 5x112 means 5 bolts with PCD equal to 112 mm.

1.1.2 Properties and Design

There are several important properties of a rim that influence both the visual appearance of the rim and also the functionality of the whole car.

One of them is the rim diameter. A larger wheel needs to be heavier and requires more energy to change its movement speed.

In terms of weight, the aim is to make the rims lighter, so that the weight of the whole car is lower, which can reduce fuel consumption as well as make the car more manoeuvrable. Lighter rims also cause less stress for the suspension system, which is absorbing forces from the wheel, for example when the car hits a bump on the road.

The rim also needs to be strong enough to maintain its shape under various conditions, such as the weight of the car it needs to carry, or the forces that emerge during the ride. It also should not easily shatter.

The shape of the center disc and spokes plays an essential role in the aesthetics of the car, but it also defines overall strength and integrity of the rim, influences aerodynamics, and can contribute to the cooling of the brakes [27].

1.1.3 Materials and Surface Treatment

These various demands, such as low weight, high strength, cost and tendency to shatter, conflict each other and influence the choice of the rim material. Among the most commonly used materials we can find steel and aluminium alloys. Steel is cheaper, stronger, but heavier, and can be enhanced by the addition of other elements. Aluminium alloys are lighter and still strong enough [27].

Occasionally also more rare materials such as carbon or titanium can be seen. These are more expensive but provide better properties. Moreover, the rim consists of several parts and each can be made from different material, which can lead to further enhancements of its features.

The surface of the rim can have various treatments such as paint, chrome, power-coating or polishing, which influence its susceptibility to scratches, oxidation, etc. They also change optical properties of the rim such as its reflectivity and color.

1.1.4 Nearby Components

When looking at the mounted wheel from the side of the car, we do not see only a rim and a tire, but there are also other components visible through the rim and around the tire, see Figure 1.4.

Through the rim, we can often see a **brake disc** and a **brake** attached to it. Above the tire, there can be parts of the **suspension system** such as a spring. This description is not exhaustive and in general, there can be more inner car components that are visible through the rim or around the tire. Furthermore, we can see the opposite wheel at certain angles and of course the surface that the car stands on, which is a conveyor belt in this scene.

1.2 Data collection

This work is based on data collected in Škoda Auto plant in Mladá Boleslav. The Škoda Auto a.s. company allowed us to install the equipment, gather the data and use them for research purposes.

Figure 1.4: Nearby Components - brake disc is marked blue, brake is red, suspension system is green, other space visible inside a car is orange and conveyor belt is violet.

The data was collected in the environment, where quality control takes place. All the cars stand still on the slowly moving belt. The area is very well lit by multiple sources. See example of this environment in Figure 1.5.

(a) (b)

Figure 1.5: Example of quality control environment in Škoda Auto. (The images come from a video on the official Škoda Auto website [52].)

1.2.1 Setup

The cameras were arranged on both sides of the conveyor belt, attached to the walls where the lights are installed. Camera A was recording cars from the right side, camera B from the left side. A top-down schema of the setup is shown in Figure 1.6. Images of the scene taken by individual cameras at the same time are shown in Figure 1.7. There is one image of an empty scene and one with a car passing by. Notice that in empty scenes, we can see the opposite cameras. Also camera B is tilted slightly downwards compared to camera A.

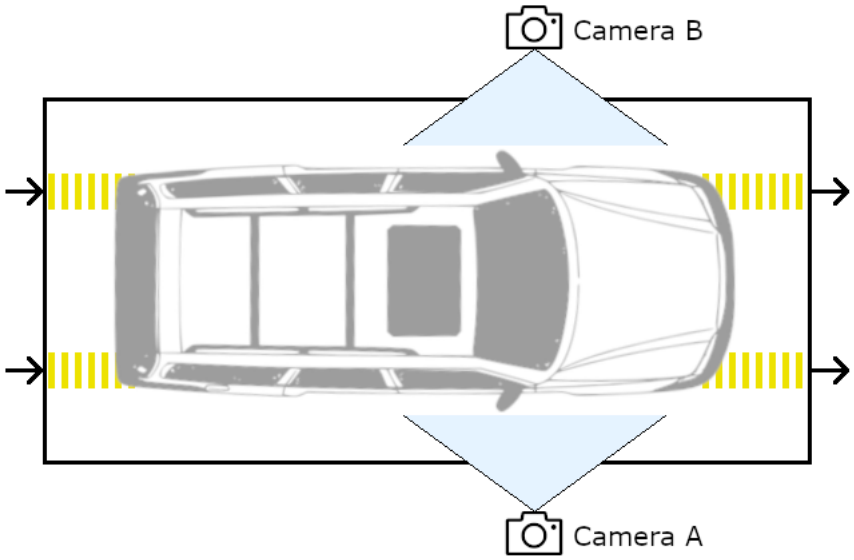


Figure 1.6: Schema of the camera setup in top-down view.

(a) Camera A with an empty scene.

(b) Camera B with an empty scene.

(c) Camera A with a car.

(d) Camera B with a car.

Figure 1.7: Examples of views from both cameras.

The hardware and a recording script were prepared in the thesis Vít [49] that was developing a computer vision system for measurement of the level of cooling liquid. They collected data in a different place of the same plant.

1.2.2 Collection

The data was collected by Adam Novozámský, who is the supervisor of both this and Vít [49] thesis. The same cameras were used, meaning Logitech BRIO 4K Stream Edition webcams. The description of the script as quoted from Vít [49, §2]:

"We wrote it as an endless cycle, that launched in two parallel threads ffmpeg.exe on both cameras simultaneously. The frame rate was set at only 1 fps and the total duration was set to 10 minutes per video. During this time all frames were recorded in motion JPEG (MJPEG) and after that encoded in h265."

1.2.3 Properties

Due to the nature of the script, the 10 minutes videos are not directly connected, but there is an unrecorded gap of approximately 20 seconds between subsequent videos. Altogether, there are approximately 204.5 hours of video recorded over the course of twelve days. For this thesis, we are using 34 hours of video, which should be sufficient sample for experiments and training of models.

From the data, we can get rough estimation of the movement speed of the belt. For the car model Octavia 3. generation, which is 467cm long [54], seen through the camera A, it takes about 40 frames from seeing the front of the car on the left side of the camera view to seeing the back of the car in the same place. Because we know the frame rate is 1 frame per second, the estimation is circa 11.5 cm/s.

Another observation is that at this speed, we can see the parts of an individual car in the scene for about 60 frames, parts of the wheel for about 15 frames and the whole wheel is clearly visible for about 10 frames. See example of sequence of frames with clearly visible rim in Figure 1.8

1.3 Challenges

The overall quality of the data is very good. We have Full HD resolution with reasonable compression that does not impact important features in the data, and the camera views capture all details that we are interested in.

Despite that, there are some characteristics that can pose a challenge. Firstly, some of the frames suffer from high brightness, which can hide important local features, such as the shape of spokes. For example see Figure 1.9. It is probably caused by the lights reflecting from the car.

Another issue can be that certain car paints are highly reflective, and therefore they can act like a mirror and reflect the wall where the camera is mounted, see Figure 1.10a.

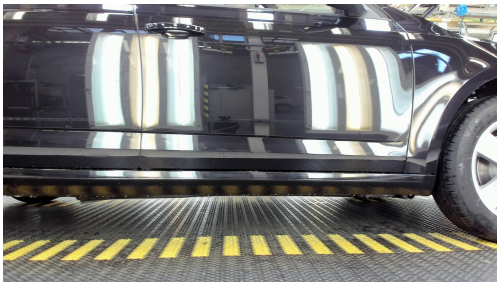
Because the data was collected during standard work shifts, there can be people and objects moving in the scene. An example of an unexpected object in the scene can be wheel of a scooter passing by, see Figures 1.10c and 1.10d.

Figure 1.8: Sequence of 12 frames from camera A illustrating conveyor belt movement speed. Car in individual images goes from left to right. Images are in sequence from left to right and from top to bottom.

(a) High brightness. (b) Normal brightness.

Figure 1.9: Comparison of frames with high and normal brightness for the same rim. Notice that with high brightness, some features of the rim are barely visible.

In terms of people, they can be seen walking, standing, sitting in the car, etc. In most cases, this does not cause any problems, but occasionally they stand in front of a rim, hiding it for the whole time it goes through the scene. See Figure 1.10b.



(a) Car paint with high reflexivity.

(b) Staff standing in front of a rim.

(c) Scooter - camera A.

(d) Scooter - camera B.

Figure 1.10: Challenges in data.

1.4 Selected Approach

As mentioned in the introduction, the goal of this thesis is to develop a proof of concept of a real-time system with the ability to automatically detect, whether all four rims on an assembled car match, meaning they are of the same type and the same size.

We decided to divide this task into three sequential tasks:

- Detection of the car and its wheels.
- Determining, whether all four rims are of the same type
- Comparison of sizes of the rims

We dedicate one chapter for each of these tasks, analyze related work and available approaches, build dataset and describe the solutions we selected, as well as their results. The advantage of this approach is that datasets and models created for these tasks may serve as resources for other works.

2. Car and Wheels Detection

The goal of this chapter is to choose and tune model for car and wheel detection. First we present related work which is split into works using only classical computer vision techniques and into works utilizing neural networks. Then we describe considered solutions in detail, create dataset and finally train and evaluate two selected models.

2.1 Related Work

The topic of car and wheel detection is well studied because it has many practical real world applications such as vehicle counting, autonomous driving, surveillance, car park monitoring, etc.

This section is divided into two parts, the first one is focused on works using traditional computer vision techniques, the latter one is concerned with the usage of convolutional neural networks.

2.1.1 Traditional Computer Vision

A great introduction to the topic of car and wheels detection is the research paper of Vinoharan et al. [47]. It introduces a three-staged approach that is able to detect the contour of a car from the side-view.

At first, they try to identify wheels using Hough transform [40] for circles. Hough transform will be described in detail in the considered methods. To eliminate false positives, meaning circle structures that are not wheels, they build a database of SURF[4] descriptors. These descriptors are, in simple terms, points of interest that can be matched to the detected circles and distinguish a correctly identified wheel from a false positive. Example of similar descriptors and their matching is shown in Figure 2.1

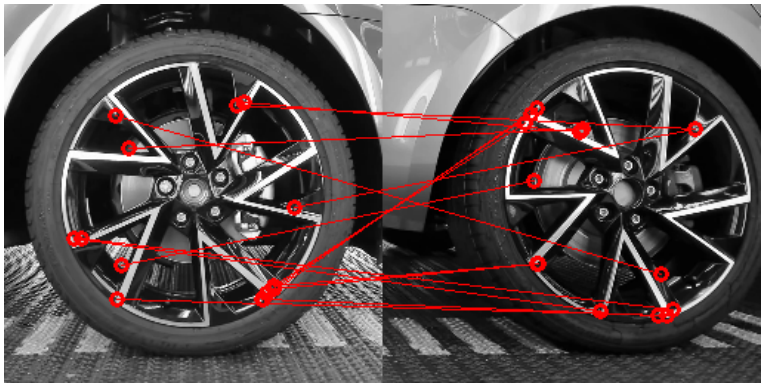


Figure 2.1: Visualization of matching of ORB descriptors, which are similar to the SURF descriptors.

In the second stage, they try to estimate a rough contour of the car using heuristics based on the wheel positions and proportions of the car. In the last stage, they use snake algorithm[25] to fit the proposed contour to the correct contour of the car.

Unfortunately, the reported results are inconclusive because their dataset is small (100 images) and consists of images of cars on white backgrounds that were collected from Google Images. Therefore, we cannot predict results of this approach in a realistic scenario.

The work of Hultström [22] uses classical methods to find wheels as well. This work utilizes real-world recordings of a four lane road from the side view.

Their strategy is to decompose the image into small windows and for each window detect, whether there is a wheel or not. The windows are first converted to feature vectors, using features such as pixel intensities and local binary pattern [22], and then they are fed to a random forest classifier [22]. The final result is then improved using heuristics and clustering techniques.

Another work Chávez-Aragón et al. [11] is focused on identifying 14 regions of interest in vehicles from the side view, such as wheels, windows, door handles, etc. The process starts with the detection of the wheels. It is performed using a classifier trained on Haar-like features [48], using Hough transform as a backup method.

Once the wheels are detected, promising search zones for other parts are proposed using methods that take into account their positions relative to the position of the rear wheel. In the promising search zones, the car parts are detected using Cascade of Boosted Classifier [48] based on Haar-like features.

The work of Grigoryev et al. [17] claims to be used in the industrial vehicle classification system deployed on toll roads in Russia. It detects wheels using fast Hough transform. Because the Hough transform is prone to detection of non-wheel circular objects, authors propose an additional method to filter out non-wheel circles in the Hough space.

These four presented works illustrate the most common approaches to wheel detection. From those four mentioned works, we can see, that for wheel detection, they rely on either Hough transform, or they extract features from a small window in the picture and then they use machine learning models to decide, whether it contains a wheel.

For a general case of traffic surveillance, we refer to overview work of Al-Smadi et al. [1]. This paper mentions important techniques used in traffic monitoring systems, for example:

- Frame differencing [41] - is a method used to detect motion in the videos. It looks for the pixels that are changing in a few consecutive frames. The change is detected by subtraction of those frames from each other - the pixels in subtraction that exceed a selected threshold are considered to be moving.

- Background subtraction [41] - is a method used to separate static background of the scene. The static background can be estimated for example by computing median of a pixel over a high number of consecutive frames.

2.1.2 Neural networks

Neural networks are machine learning models, which are able to both extract features and perform the required task, such as classification or object detection.

In this thesis, we will focus on a specific category of neural networks - deep convolutional neural networks [26]. These networks use convolutional layers, which are suitable for image data processing, and achieve state of the art results in many computer vision related tasks.

The work of Song et al. [43] uses convolutional neural networks (CNN) to perform vehicle detection, tracking and vehicle counting from highway camera recordings. The paper provides useful overview of traditional methods used in vehicle tracking, as well as reference works, that use neural networks for that purpose.

First they perform road segmentation to eliminate environmental influences. For vehicle bounding box detection they decided to use YOLOv3 network [32] for its speed, high accuracy and the ability to detect objects of various sizes. For tracking, they use ORB algorithm [36] to extract features of the vehicle in the bounding box and compare it with near vehicles in the next frame.

2.2 Considered Methods

Based on the presented related work we selected several common approaches suitable for car and wheel detection. In this section we describe them in more detail and discuss their advantages and disadvantages.

2.2.1 Hough transform

Hough transform is a well known method for detection of geometric shapes that can be defined by a small number of parameters, for example lines or circles. That is why it is suitable for detection of rims or wheels. To illustrate the main idea of the algorithm, we will describe what the process looks like for circle detection.

A circle in the image can be defined by three parameters, for example two coordinates of the center and its radius. Each of those parameters can represent one dimension in a three-dimensional space. A point in this space can be mapped to a circle with a given center and a radius.

We can discretize this space, for example to integer values, and limit each dimension to a reasonable interval. For example coordinate values would be from zero to maximal width/height of the image. This will leave us with a finite three dimensional array of cells, where each cell can represent one set of circles.

We can iterate through all the cells and for each cell (and the circle it represents) check, whether there is enough evidence that this circle is present in the image. Such evidence can be for example the number of edge pixels in canny edge detection [10] present in the area of the inspected circle.

We will consider the cells with the largest values in the array, or cells that pass a certain threshold and declare them to be circles in the image.

The downside of this algorithm is that the space and time needed grows exponentially with the number of parameters of the shape. Furthermore in practice, the algorithm is prone to finding false positive circles in unexpected places, that is why the parameters of the array need to be carefully tuned or there needs to be another procedure that would eliminate these false positives.

On the other hand, the algorithm provides simple explanation how shapes were found and is fast enough for real-time processing. There are several versions

of the algorithm [42] that improve its speed and memory consumption.

2.2.2 Window classification

The fixed window classification is based on the premise that we can expect certain objects to be found in certain parts of the image. For example in our case of rim detection, we can imagine a window in the center of the image, which will be the size of the largest rim and we can expect that the rim will go through this window in one of the frames, as it is moving through the scene.

We can extract features from this window and pass them to a machine learning classifier to decide whether the window contains rim.

This procedure is fast, but can encounter problems when the rim is covered by another object exactly when it should appear in the window. The solution would be to use multiple windows in different locations where we expect rims, or use a sliding window, that would go through the whole image. This would slow down the detection, but ensure that no parts of the image are missed.

The general problems of these methods are scales and aspect ratios of the objects in the images, which can be solved by running the sliding window on downscaled versions of the images. In our data, both scale and aspect ratios are well known, so it would not be an issue.

2.2.3 Object detection CNNs

Another suitable approch to detecting rims and wheels are object detection networks. Here we present three popular architectures.

R-CNN

The first one is the family of R-CNN networks. It consists of R-CNN [16], Fast-RCNN [15] and the most advanced Faster-RCNN [34]. The main idea is to divide object detection into four independent tasks [13]:

- Extraction of regions of interest - algorithm, or part of the network, that is responsible for selecting parts of the image, that seem to contain an object.
- Feature extraction - is usually a pretrained CNN that extracts features from the raw image.
- Classification module/head - algorithm, or part of the network, that classifies region of interest based on its features.
- Localization module/head - algorithm, or part of the network, that produces tight bounding box for an object.

In comparison to its predecessors, Faster-RCNN has all modules implemented in one end-to-end trainable neural network. This makes it much faster in both training and inference and improves the accuracy. In general, this architecture provides high accuracy, but is slower than the following two models.

SSD

The next model is the Single-shot detector [28], SSD in short. It is a single-stage detector, which means that it does not perform region proposal and then classification separately as the R-CNN family, but does both at once. This makes it significantly faster.

The architecture consists of 3 stages [13]:

- Feature extraction - handled by a pretrained CNN that extracts features from the raw image.
- Downscaling feature layers - performed in convolution layers that enable detection on various scales.
- Non-maximum suppression - ensures, that there will be only one bounding box for one object.

YOLO

The last family of models is called You Only Look Once, shortly YOLO. The first three versions v1 [33] , v2 [31] and v3 [32] were presented by original authors. In 2020, two more versions were introduced by different authors. They called them similarly: v4 [7] and v5 [23]. The YOLO architecture is popular because its speed enables its usage on mobile devices, such as smartphones.

The main idea [13] is to divide the image into a square grid where each cell is responsible for prediction of a given number of bounding boxes, the probability that they contain an object and the object class. Similar to SSD, YOLO utilizes a pretrained CNN for feature extraction and uses convolution layers of different sizes to detect objects of multiple scales.

A problem resulting from the design of the architecture is that if more than the expected number of objects were present in one cell, the network would not be able to detect them all. However this should not be a problem for our use case, as we are detecting a small numbers of relatively large objects.

2.2.4 Segmentation CNNs

The last approach we present are segmentation CNNs which provide not only boundaries and the class of the detected object, but it provides also a mask that specifies the object location at the pixel level. Representative architectures for this task may be Mask-RCNN [19] and U-net [35].

Mask-RCNN is an architecture that adds a mask generation head on top of Faster-RCNN. U-net is a fully convolutional architecture originally developed for precise biomedical image segmentation, but it was already adopted to wide range of segmentation tasks.

The downside of image segmentation is that the labelling of custom data is more time consuming than in case of bounding boxes, even when using semi-automatic tools that provide a pixel mask proposal based on pretrained machine learning models.

2.2.5 Selected approach

In this thesis, we have decided to experiment with Hough transform and YOLO neural network. Hough transform should give us baseline model for wheel detection and is fast enough for real-time evaluation. YOLO is expected to provide bounding box detection of cars and wheels with high accuracy because our task with only two classes and stable environment should not be hard to learn. The second reason for YOLO is fast inference that enables the system to react in real time and on top of that leave remaining computing power for tasks that will be discussed in the following chapters.

2.3 Data preparation

Training a neural network for object detection requires providing training data where the objects to be detected are indicated and correctly labeled. We decided to manually label the wheels and cars in the training images using bounding boxes. Bounding box is an axis-aligned rectangular shape, that tightly contains the object. This approach was selected because the labelling procedure is much faster than marking polygons, or pixel masks, and is sufficient for our needs - detection of position and size of the object.

2.3.1 Computer Vision Annotation Tool

To label the data, we used Computer Vision Annotation Tool by Sekachev et al. [39], also known as CVAT. It is a mature, free, open-source tool built on Django web framework. The used version of core was 3.13.3.

It supports multiple labeling options, such as rectangles, polygons, polylines, points, etc. It is also able to export the labeled dataset into common formats such as COCO, ImageNet, PASCAL VOC, TFRecord, YOLO and many more.

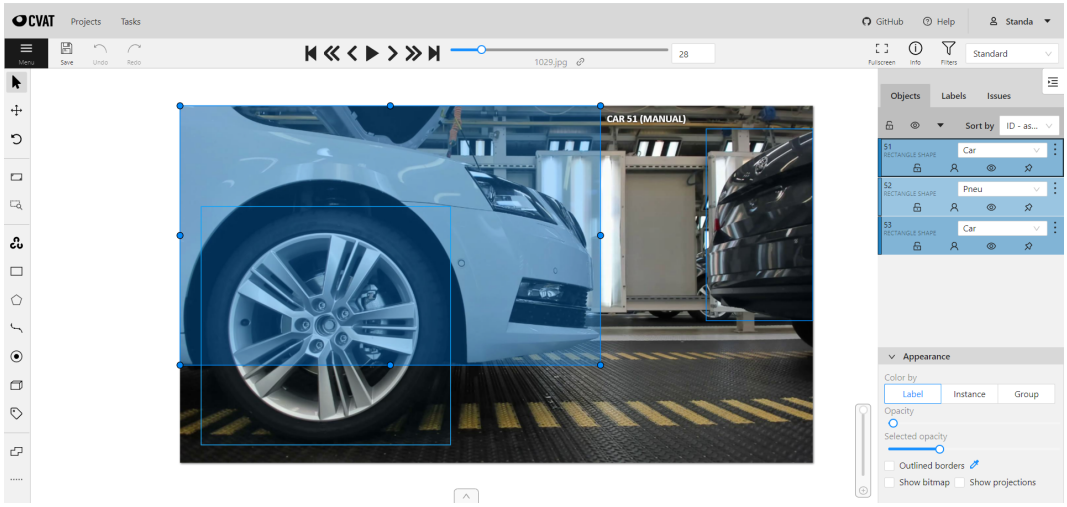


Figure 2.2: Computer Vision Annotation Tool - User Interface.

2.3.2 Dataset

The dataset consists of 1000 training frames, 250 validation frames and 250 testing frames. It can be used either as an input of a machine learning model, or for tuning parameters in different environmental conditions in case of algorithms like Hough transform.

To create the dataset, we split the collected data (videos) in the order they were recorded into three sets in ratio 2:1:1. From the first set we randomly selected 1000 frames without repetition for training. In the similar way, we created validation examples from the second set, and test examples from the third set.

In the dataset, an object was labeled a car every time we recognized it as a car part, that is not a wheel. If the car was visually split into multiple parts by another object (passing person), we label all the visible car parts with the same bounding box. A wheel was labeled only if it was recognizable by human, meaning it was present in the scene and not mostly overshadowed by other objects. The dataset also contains images with no labels, for example training set contains 91 images of this kind. These images help the algorithm to avoid false positive detections.

2.4 Experiments

At first we tune Hough transform as a baseline model to see results that can be reached with a relatively simple procedure. Afterwards YOLOv5 is trained to detect both cars and their wheels and achieve acceptable results that can be used in practice and run in real-time.

Setup

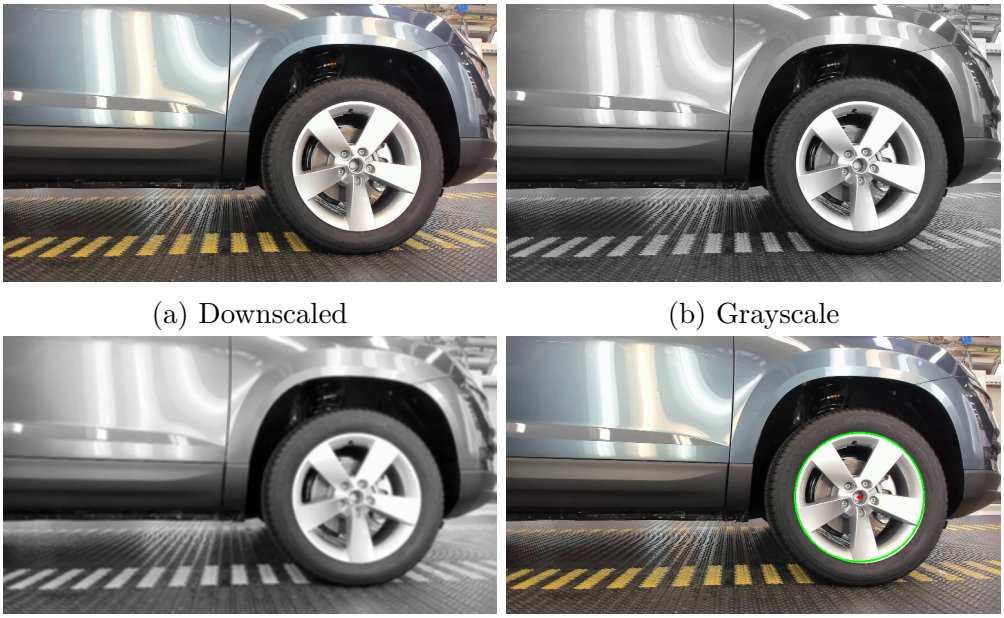
To give context to time measurements such as train time and inference time of models, the following hardware setup was used: Dell G5 15 notebook equipped with Intel Core i7 10750H Comet Lake processor and NVIDIA GeForce RTX 2070 Max-Q 8GB graphic card. The same hardware was used throughout the whole thesis.

Hough transform

For Hough transform, we used an implementation provided by openCV [8], which is a variant of Hough transform called 21HT [51] for circle detection, that is optimized to minimize memory consumption.

To improve its properties, we performed several preprocessing steps. First, the image is downscaled to reduce both time and space requirements of the algorithm. Then it is converted to grayscale, which is the required input format for Hough transformation. Finally the image is blurred using a Gaussian blur to reduce noise and erase high frequency components, that can lead to false positives. All these steps are visualized in Figure 2.3

To tune parameters of the preprocessing pipeline and Hough transformation, we implemented a simple tool where the user can go through the training part of the dataset, view images in the individual steps of the pipeline and easily change the parameters. Although there are possible approaches for automatic



(a) Downscaled (b) Grayscale

(c) Blur (d) Hough transform

Figure 2.3: Hough transform preprocessing pipeline

selection of Hough transform parameters, we performed the parameter tuning manually because it was less time demanding and the reached accuracy is not so crucial since the Hough transform will serve only as a baseline model. During the parameter tuning, we mainly focused on eliminating false positive detections.

The results measured on validation data are summarized in Table 2.1 and they are satisfactory for a baseline method.

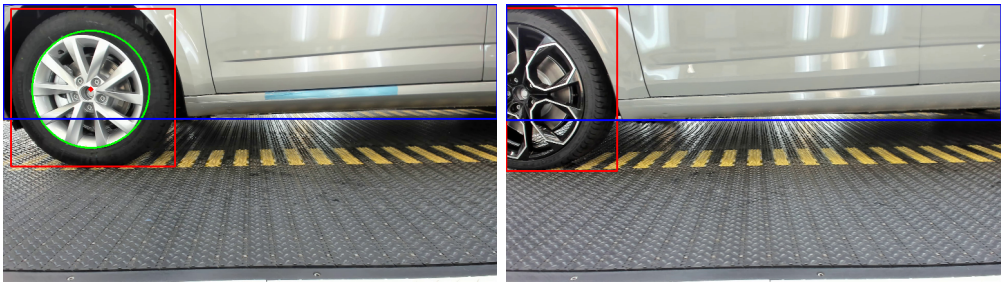
True positives	128
False negatives	54
False positives	0

Table 2.1: Results of Hough transform on validation data

The true positives are not always perfect contours of the rim. This is caused by the fact that the contour of the rim in the image can also be elliptical because of the rotation of the wheel to the camera. See example in Figure 2.4a.

The 54 reported false negatives can be split into three groups. In the first group of roughly 20 samples, the rim was clearly visible and we expected detection but it failed. Those were mostly black rims and rims with small part missing due to the edge of the image. Example is presented in Figure 2.4b. The second group of about 10 samples are rims with more than half of the rim area missing. Those detections would be very hard for Hough transform anyway and should not be considered as a detection mistake. The third group of size around 20 are wheels, where the rim area is not visible, but we can see a part of tire. This group is the hardest and it also should not be considered as a failure.

The rough estimation of runtime of the preprocessing and Hough transformation on a single frame is circa 8 ms which is excellent for real-time processing.



(a) Elliptical rim contour (b) Partially visible dark rim

Figure 2.4: Problematic cases for our Hough transform detection

YOLOv5

To train YOLOv5 [23], we used Github repository from the original authors. The repository contains models with pretrained weights on COCO val2017 dataset and training scripts with a support for augmentation.

In order to present our results, first we need to introduce the mAP metric [13], which is used to evaluate object detection models.

The data is labeled as bounding boxes with class of the object they contain. Models predict bounding boxes, the class of contained object and its probability.

To say that the predicted box is a good fit for the golden box, we will use a numeric value called Intersection over union. It is computed as the area of their overlap divided by the area of their union. To use it in practice, we need to set a threshold value that specifies a good fit, for example .5, which means, that the intersection of the boxes needs to be at least 50 percent of their combined area.

For a model, we can also measure precision and recall. Precision computes how many of our detections were correct. Recall computes how many golden data we detected. To influence precision and recall of a model, we can introduce a threshold to the probability of model detections. That will consider only detections with probability equal or larger than threshold. Going through all the values of the threshold and plotting precision and recall to a graph, we obtain a precision-recall curve. This curve can be plotted for individual classes of object and provides an insight into the behavior of the model.

On the precision-recall curve, we can calculate average precision, shortly AP, either by calculating the area under curve, or by sampling its value in equidistant values of recall, such as 0.1, 0.2,... 1.0. By calculating mean of AP over all classes, we get mean average precision, shortly mAP. In practice, we can see mAP written together with a number, such as mAP@0.5, which defines the intersection over union threshold we are using. mAP@.5:.95 then means average of mAP over interval of intersection of union ranging from 0.5 to 0.95 with step value 0.05.

Going back to the training, we have selected two small models from the YOLOv5 family made for mobile devices. The first one is YOLOv5s, which has 7M parameters, the latter is YOLOv5n with 1.9M parameters. Their size should be both sufficient for this task and small enough to leave resources for the necessary follow-up tasks described in the next chapters.

The training was done for 50 epochs starting with the default parameters in the repository that are documented in Attachment A.1. The results on validation data for YOLOv5s are in Table 2.2 and for YOLOv5n in Table 2.3. In both cases,

the models have successfully predicted almost all data with .5 threshold, and were fairly successful with higher thresholds.

Estimation of inference time on a single frame for YOLOv5s is circa 26 ms which is suitable for our application.

Class	**Labels**	**Precision**	**Recall**	**mAP@.5**	**mAP@.5:.95**
Wheel	182	0.983	0.97	0.993	0.962
Car	300	0.983	0.98	0.993	0.928
All	482	0.983	0.975	0.993	0.945

Table 2.2: Results of YOLOv5s on validation data

Class	**Labels**	**Precision**	**Recall**	**mAP@.5**	**mAP@.5:.95**
Wheel	182	0.978	0.989	0.99	0.952
Car	300	0.993	0.966	0.988	0.938
All	482	0.986	0.978	0.989	0.945

Table 2.3: Results of YOLOv5n on validation data

Discussion

Hough transform as a baseline model performed very well. Because we have labeled all wheels recognizable by a human, most of the reported false negatives should be considered too hard for this method and we see much better results when at least a half of the rim area is visible. In case the false negatives would cause problems and leave undetected wheels, the parameters can be tuned to minimize false negatives which would cause rise of false positives, that can be filtered out by an additional method.

YOLOv5s provides high accuracy and sufficient inference speed and we will be using it further in this work. It will play role in data preparation for chapters 3 and 4 and furthermore it will be part of the pipeline of the final system. We believe that this model is flexible enough to cope with slight changes in environment and differences in cars and wheels. In case there would be a major drop in accuracy, the model can be trained on new data. It should not be a problem in practice, as training on our machine took about 100 minutes.

3. Rim classification

In this chapter, we will use wheel detection model from the previous chapter to build a database of rims and split them into classes based on the differences in their shape and color.

Because we were not able to find related work that is performing rim or wheel classification, that is why we start this chapter with a description of classification techniques which can be used. Afterwards we continue with creation of datasets and in the end we experiment with selected models.

3.1 Considered Methods

3.1.1 Traditional Computer Vision

In traditional computer vision techniques we will introduce an approach that uses combination of suitable feature extraction and machine learning classifier. This approach was already mentioned in the related work in the second chapter where Hultström [22] combines features such as pixel intensities and local binary pattern with a random forest classifier.

We decided to try combination of histogram of oriented gradients [20] feature extraction technique, shortly HOG, together with support vector machines (SVM) classifier. We will not go deeper in description of SVM because we consider it outside of the scope of this thesis and we will mainly focus on HOG because it is an interesting method for description of shapes in the image. Moreover the number of features that reasonably describe overall geometry of the image can be quite small.

Before we introduce the main idea, we need to present the definition of the pixel gradient. A pixel usually has 4 neighboring pixels with whom it shares its edges. By computing the difference between its right neighbors value and left neighbors value, we get number that we will call gradient in horizontal direction. Similarly, by computing the difference of the top and bottom neighbors, we get gradient in vertical direction. If we combine gradient in horizontal direction and gradient in vertical direction, we get a two-dimensional vector that will be called gradient of the pixel. We can also represent this vector by its angle and magnitude.

Now we split the image into a grid of square cells where each cell will be 8 pixels per side. The goal is to represent this cell of 64 pixels by 9 numbers that we will call histogram of gradients. Each of these numbers are representing one of the angles: 0, 20, 40.. 140, 160 degrees. Notice that those angles represent only half of the circle. It is because we are not interested in directions of the vectors.

For each pixel, we take its angle and add its magnitude to the nearest angles in the histogram according to the distance in degrees. For example if we have pixel with angle of 25 degrees and magnitude 10, we will add 7.5 to the 20 degree value in histogram and 2.5 to the 40 degree value in histogram.

This will provide us with histogram of the most important angles in the cell. We can see visualization of these histograms in Figure 3.1.

(a) Original image (b) Histogram of oriented gradients

Figure 3.1: Visualization of histogram of oriented gradients feature extraction method - we can see individual cells, each with a histogram in its center.

HOG can me improved to be more resistant to changes in brightness by averaging histograms over several cells. We call the unit of this averaging a block.

3.1.2 Neural networks

In the last ten years, the deep learning community comes every year with new ideas and models that greatly increase accuracy on the datasets compared to their predecessors (especially in the task of image classification). For overview of this development and the most important milestones, we refer to the survey of Alzubaidi et al. [2].

In this section, we will focus on description of transfer learning and one of the currently best performing classification network EfficientNet.

The transfer learning [2] works with the idea that if we have a smaller dataset, we can take a model pre-trained on large dataset, for example ImageNet [37], and use its learned representations to train our dataset quicker and better. It is done by making the last few layers trainable and the rest of the network not trainable, in jargon those layers are called frozen layers. The number of layers we want to unfreeze for training differs based on the used model, dataset size and complexity.

In terms of EfficientNet, its authors Tan and Le [44] examined previous approaches to model scaling depth, width and resolution of the network and they came up with a novel way how to do it more efficiently and achieve better results.

Furthermore they have proposed a new model called EfficientNet, that was found using neural network search and they scale it by the technique they have developed. It is based on mobile inverted bottleneck MBConv from the MobileNetV2 [38] and squeeze-and-excitation optimization [21]. See the description of the smallest EfficientNet-B0 baseline network in the Figure 3.2

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Figure 3.2: EfficientNet-B0 baseline network - table taken from the original paper [44].

3.2 Data preparation

The goal of data preparation in this chapter is to create datasets of rims for an image classifier. We decided the format to be square images of wheels. The resolution we are using is 256×256, which is smaller, than the original image, but still contains fine details of a wheel.

3.2.1 Preprocessing

At first, we went through the data and we have estimated, that the largest wheel in the dataset fits roughly into a square of size 770×770. Afterwards, the YOLO object detection network from the previous chapter was utilized. We ran object detection inference on every frame. For each bounding box with class "wheel", we calculated its center and extracted a cropped image of size 770×770 with center in the center of bounding box.

In case part of this cropped image was out of the frame, we moved it to the border, so it would fit into the image. Each of these cropped images were then scaled down to 256x256 using bicubic interpolation.

3.2.2 Classes Overview

As we went through all extracted images of rims, we have identified 31 unique rim classes that have differences in either shape of the rim, color of the rim or both. We tried to hand pick at least 300 images of each class. In most cases it was possible, but there were 10 classes that had less than 300 images in the whole dataset. For visual overview of classes see Figure 3.3.

The procedure for image selection was that these images does not necessarily need to have a whole wheel visible, but the class has to be identifiable by a human. It usually means that at least half of the wheel needs to be visible in the image.

The labels are numeric values and the idea behind them is that every unique shape was labeled as multiple of ten, for example label '140', and color variants are defined by addition of a single digit number, see label '141'.

From the selected images were created two datasets.

Figure 3.3: Overview of all rim classes - images of individual classes have label and sample count below themselves. The classes where we acquired less than 300 samples are highlighted by a red rectangle.

3.2.3 Dataset21

In this dataset every class has 100 training samples, 25 validation samples and 25 test samples (ratio 4:1:1). Because not every class has enough samples for the selection procedure we are using, we decided to include only classes where we had 300 and more samples. Only 21 classes satisfy this condition, that is why the dataset is named Dataset21.

The procedure to create the dataset was following: in every class the samples were ordered by time and then split into 3 groups in ratio 4:1:1. From the first group the 100 training samples were randomly selected, from the second group 25 validation samples were randomly selected and from the third group 25 test samples were randomly selected. The important note is that we did not want to have samples from one car in both training and validation set, or in both validation and test set, that is why the sizes of the group were not exactly 4:1:1 but we manually ensured that the dividing place between the sets is clearly between two cars.

The detection of the wheel in the previous chapter can make errors and produce bounding boxes that are false positives, meaning they are labeled as "wheel" but in fact they do not contain any. For that reason we have added one more class 'other' to the dataset. It consists of cropped images that do not contain wheel, or they do, but it's mostly hidden behind other objects, thus we can't recognize which class it is.

In the sum, the dataset has 2100 training samples, 525 validation samples and 525 testing samples.

3.2.4 Dataset31

This dataset consists of the previous dataset plus the 10 classes that were left out in the previous Dataset21.

For classes that have less than 150 samples, we took all the data ordered by time and split them into train/val/test sets in ratio 4:1:1. For classes with more than 150 samples, we again split the data into three groups in ratio 4:1:1 and randomly selected 100 training samples from the first group, 25 validation samples from the second group and 25 test samples from the third group.

Because those classes have low count of samples, we cannot enforce that one car, or even one wheel, does not appear in more than one set. The only guarantee is that even if one wheel is used in more sets, the samples will always come from different frames.

In the real production, one would be able to acquire enough data for every class to avoid this data leakage. We incorporate this mainly to test how will the models behave with higher class count and low number of samples in some of them.

Similarly to the previous dataset, the 'other' class is also added to this dataset. In the sum, the dataset has 2816 training samples, 711 validation samples and 711 testing samples.

(a) Original (b) Coarse dropout (c) Rotate

(d) HSV change (e) RGB shift (f) Brightness, contrast

(g) Motion blur (h) Optical distortion (i) Combination of all

Figure 3.4: Rim dataset augmentation

3.2.5 Augmentation

In order to support generalization, we decided to use data augmentation [9]. Types of applied augmentations are presented in Figure 3.4.

The coarse dropout acts like objects that can appear in front of the rim. Rotation augmentation simulates slight rotation of the rim. It enhances data because even though in the original dataset a single rim can be sampled multiple times, it has the same rotation every time, as the car is standing on the conveyor belt.

HSV and RGB changes modify colors of the car and the belt. Brightness and contrast changes simulate bad lightning conditions, that sometimes appear in the original dataset.

Motion blur mimic the blur caused by movement or by the loss of camera focus. The last used augmentation is optical distortion, which is a slight modification of image geometry.

3.3 Experiments

3.3.1 Histogram of Oriented Gradients and SVG

In this experiment, we are going to measure accuracy of hog features and svm classifier. We conduct multiple experiments with various hog parameters.

Setup

The selected dataset for this is the easier Dataset21. For HOG we set as a constant parameter 'multichannel' which indicates, that we are working with color image, hence we select the gradient for the pixel from the color channel where it is largest. The parameter 'cells_per_block' is also set as a constant. It describes the number of cells over which we normalize the histogram.

On the other hand, multiple values for parameters 'orientations' and 'pixels_per_cell' are tested. The 'orientations' parameter defines, how many bins does histogram for one cell have. Parameter 'pixels_per_cell' describes the size of one cell in pixels.

The SVM variant used is LinearSVC from the scikit-learn library [30].

Results

The individual runs with tested parameters and results are in Table 3.1. To paint a picture of memory size of features for a single image, one feature is represented by float64 meaning 64bits per feature. Thus for example 100K features take 800kB of space.

The confusion matrix for the best run is displayed in Figure 3.5.

Orientations	**Pixels per cell**	**Val acc**	**Train time [s]**	**Features**
9	(4, 4)	0.542	558 + 812	331K
9	(8, 8)	0.643	165 + 378	73K
9	(16, 16)	0.706	64 + 137	16K
9	(24, 24)	0.697	47 + 52	5K
13	(4, 4)	0.539	550 + 922	450K
13	(8, 8)	0.649	172 + 377	105K
13	(16, 16)	0.704	75 + 161	23K
13	(24, 24)	0.744	47 + 56	7.5K
16	(4, 4)	0.537	567 + 802	553K
16	(8, 8)	0.666	171 + 352	130K
16	(16, 16)	0.712	69 + 149	28K
16	(24, 24)	0.718	56 + 66	9.2K

Table 3.1: Results of HOG with with multiple variations of parameters. Values in column 'train time [s]' consist of two values, time to generate HOG features from images and time to train SVM. The column 'features' describes the number of features generated per image.

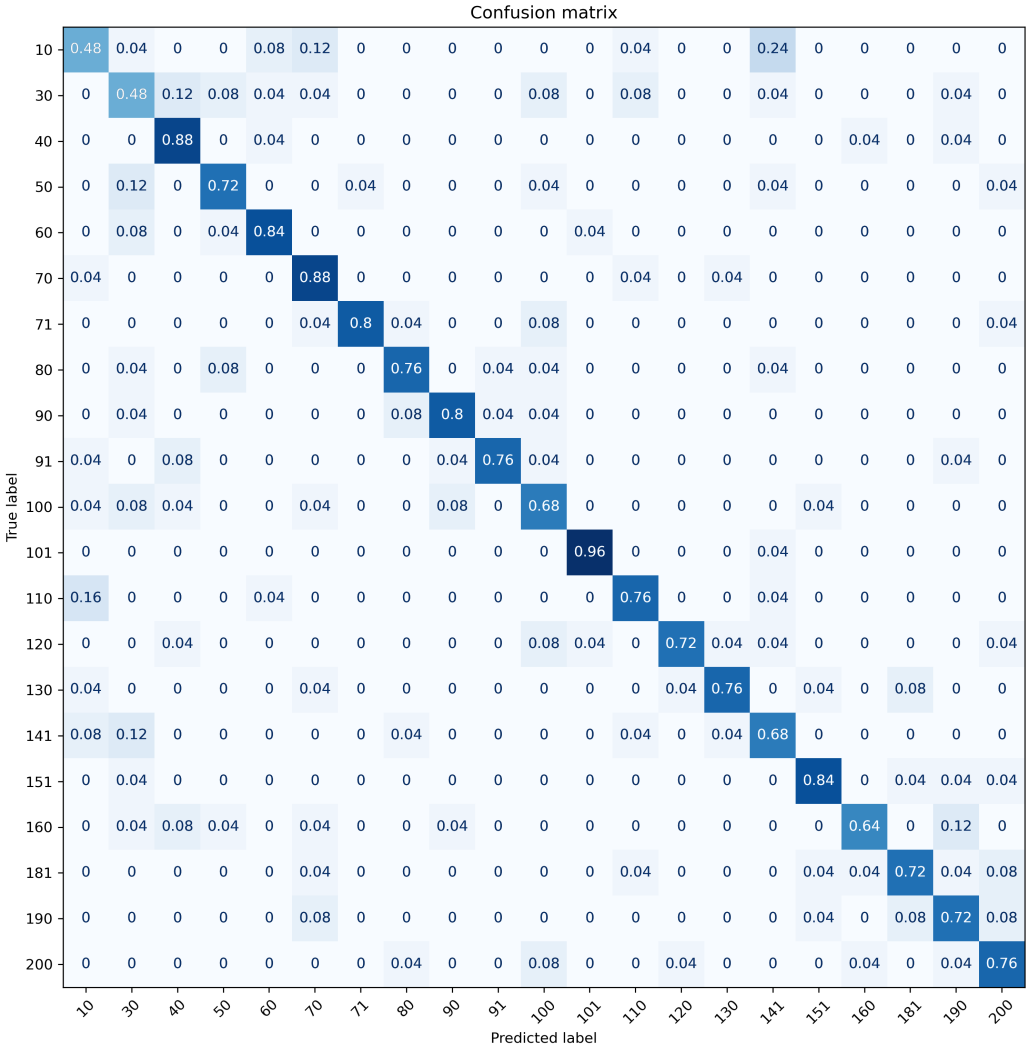


Figure 3.5: Confusion matrix for HOG experiment with orientations = 13 and pixels per cell = (24, 24).

Discussion

The highest achieved validation accuracy is 74% which is probably too low for practical use, but can serve as a baseline for further experiments.

From the results we can see, that accuracy is usually high with larger cells, which also brings benefit of lower training time and less features per image. The confusion matrix does seem to have more or less random misclassifications, we can only point out low accuracy in classes '10', '30'.

The training time is sufficient for practical use. The inference time grows with the number of features and ranges between 1 ms and 7 ms, the time to convert image to HOG features also grows with the number of features and ranges between 20 ms for the smallest and best achieving runs and 200 ms for the runs with around 500K features.

3.3.2 Unfreezing layers of EfficientNet

The goal of this experiment is to determine how many layers of EfficientNet should we train and how many layers should we leave frozen to achieve optimal results with transfer learning.

Setup

The selected dataset for this experiment is Dataset21.

The selected model is the smallest proposed model in original paper EfficientNetB0. Without a classification head, it has 237 layers that altogether contain 4M of parameters. The weights are pretrained on ImageNet dataset. For our experiment, we have only added our own softmax layer suitable for our dataset.

The training was set for 50 epochs because the models seem to converge without problems for this setting.

Results

Individual runs with number of unfrozen layers and achieved validation accuracy are in Table 3.2 and plots of train and validation accuracy throughout the runs are in Figure 3.6.

Unfrozen layers	**Trainable params**	**Max val acc**	**Train time** [s]
0 (only head)	0	0.948	500
1	0	0.954	593
2	2.5K	0.946	512
3	412.2K	0.961	529
5	781.4K	0.984	551
10	893.2K	0.984	529
25	1.5M	0.996	605
50	2.5M	0.996	656
100	3.5M	0.996	843
237	4M	0.995	1932

Table 3.2: Results of runs with various number of unfrozen layers.

Discussion

The results suggest that we can train an almost optimal model by unfreezing twenty five or more layers. Also from this number of layers, we can see nice convergence of training and validation accuracy curves. In terms of training time, most of the runs take less than 11 minutes and we can see slight increase to 14 minutes with 100 unfrozen layers and to 32 minutes with the whole model unfrozen.

The inference time around 60 ms per image which is promising for our application.

3.3.3 EfficientNet on Dataset31

In this experiment, we test EfficientNet on Dataset31 to see its performance on higher class count and how it deals with a low number of samples in some classes.

Setup

The training is performed with 25 unfrozen layers for 50 epochs. Furthermore we use class weights for training the model in order to compensate imbalanced classes. The weights are computed by following formula:

$$class\_weight = \frac{100}{number\_of\_samples\_in\_class}$$

It will assign weight of 1 to classes with full number of 100 training samples and higher weight to classes with less samples. The samples with higher weight influence model training more a it should lead to better generalization.

Results

Maximum validation accuracy for this run is 0.973. The training plot is in Figure 3.7 and confusion matrix in Figure 3.8.

Discussion

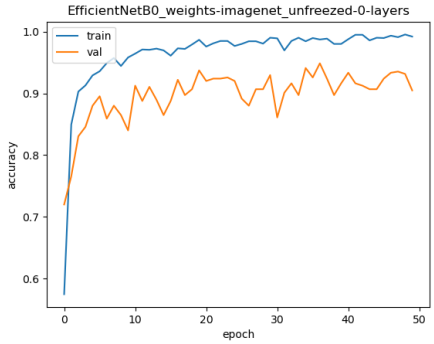
Validation accuracy is worse than in the case of Dataset21. From the confusion matrix is clear that the most of misclassifications happen on the classes with low sample counts.

3.3.4 Augmentation and EfficientNet

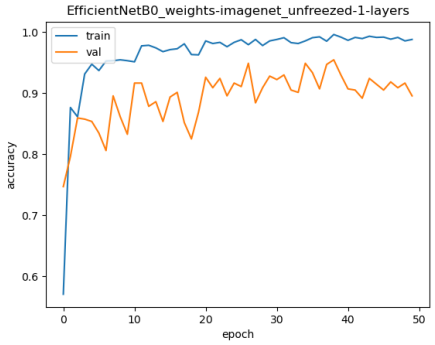
In this experiment we try to improve accuracy of the EfficientNet model on the Dataset31 by adding augmented data to the training.

Setup

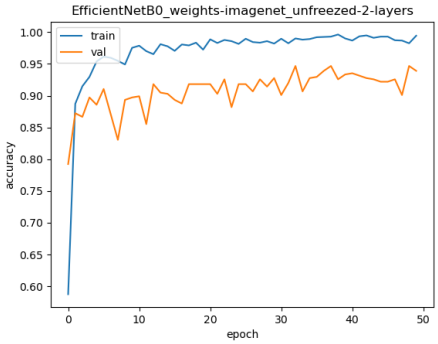
The training is performed with 25 unfrozen layers on Dataset31 for 50 epochs. To each class we add 50 augmented images. Those images are generated by randomly selecting image from class a running augmentation on it. The class weights



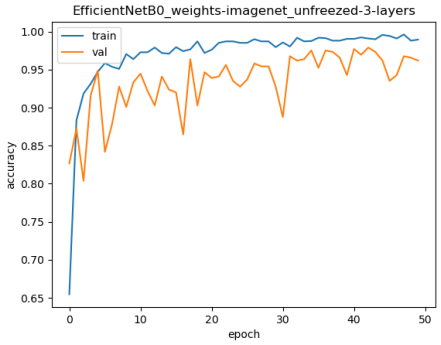
(a) 0 unfrozen layers



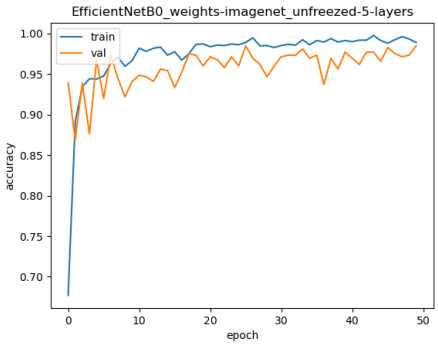
(b) 1 unfrozen layers



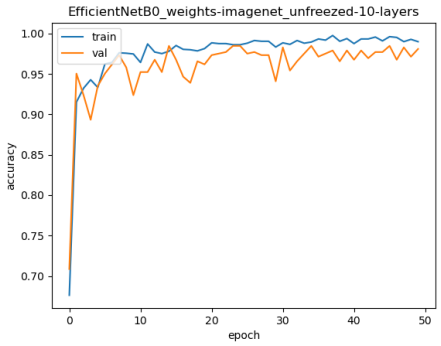
(c) 2 unfrozen layers



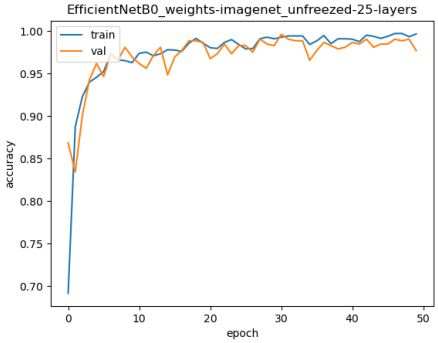
(d) 3 unfrozen layers



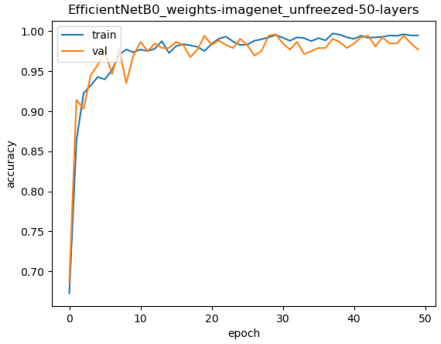
(e) 5 unfrozen layers



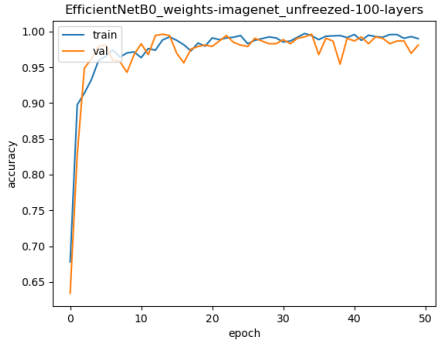
(f) 10 unfrozen layers



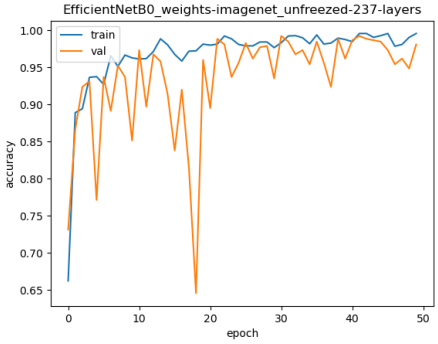
(g) 25 unfrozen layers



(h) 50 unfrozen layers



(i) 100 unfrozen layers



(j) 237 unfrozen layers

Figure 3.6: Train plots for various numbers of unfrozen layers - x-axis is number of epochs, y-axis is models accuracy, where blue line represents training accuracy and orange line validation accuracy. We can observe that the most stable training has models with unfrozen 5, 10 and 25 layers.

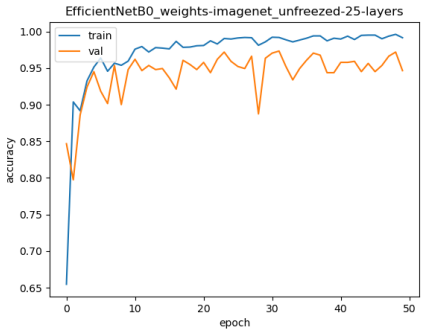


Figure 3.7: Training plot for EfficientNet and Dataset31 experiment

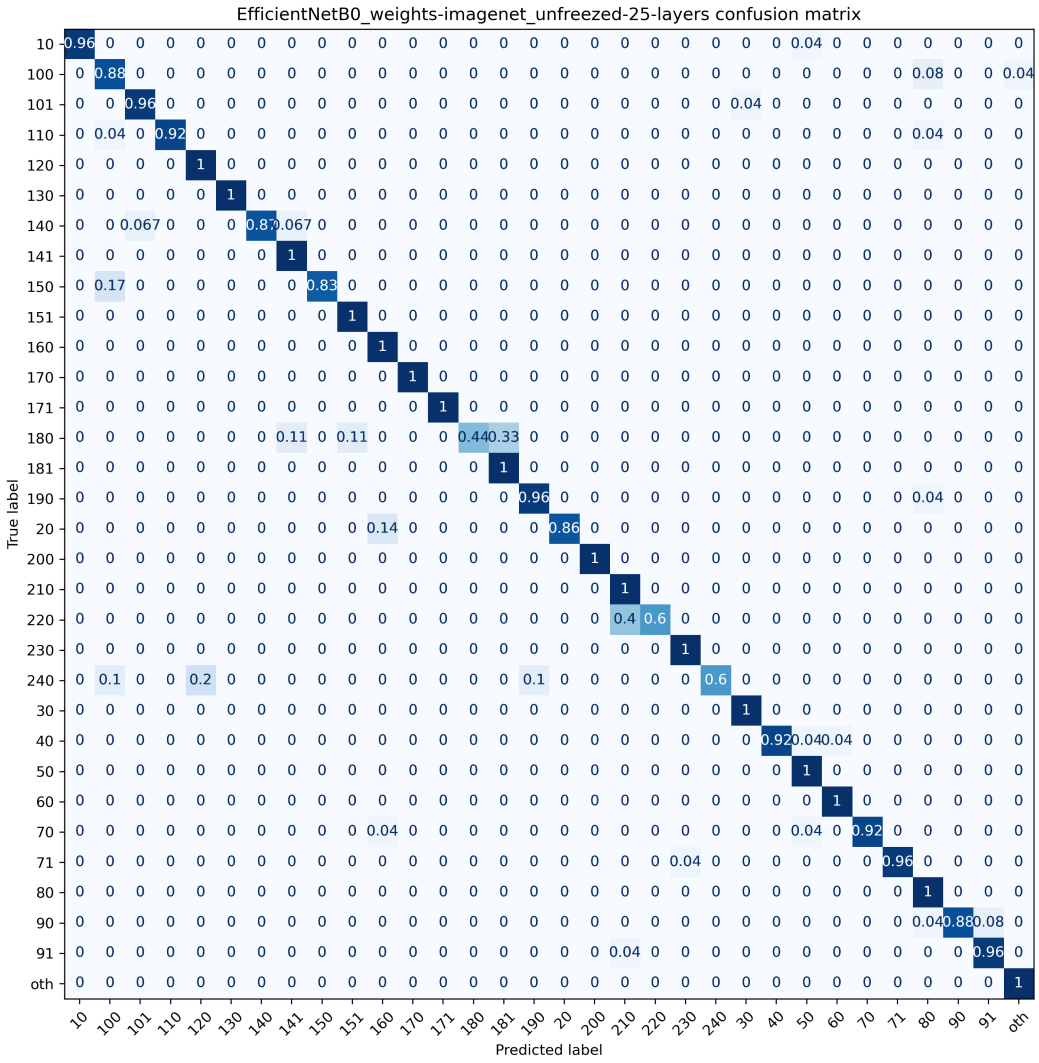


Figure 3.8: Confusion matrix for EfficientNet and Dataset31 experiment.

formula is similar as in previous experiment and accounts for those additional images:

$$class\_weight = \frac{100 + 50}{number\_of\_samples\_in\_class + 50}$$

Results

Maximum validation accuracy for this run is 0.975. The training plot is in Figure 3.9.

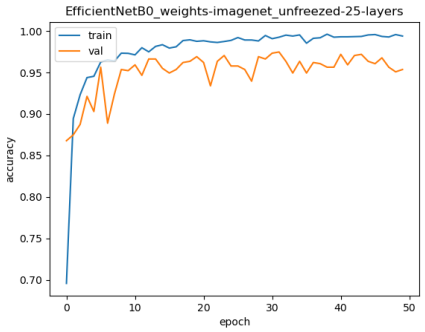


Figure 3.9: Training plot for EfficientNet and Dataset31 with augmentation experiment

Discussion

The augmentation does not seem to substantially improve validation accuracy of the model.

3.3.5 EfficientNet Evaluation on Test Set

The last experiment presents accuracy of selected model on the test set.

Setup

For this experiment, we use EfficientNet with 25 unfrozen layers trained for 50 epochs. There will be no data augmentation because it did not seem to provide any advantage in the previous experiment. The performance will be tested on both datasets.

Results

Maximum test accuracy on Dataset21 is 0.986 and for dataset 31 is 0.973. For Dataset31 can be also mentioned balanced accuracy that equals 0.949. The balanced accuracy we are using is provided by scikit-learn library and is computed as "average of recall obtained on each class" and plays the same role as class weights during training, giving all classes the same importance by promoting the effect of samples from classes with small training set.

3.3.6 Discussion

The combination of HOG features and SVM classifier provides a good baseline model, although it does not reach results expected for practical use. It would be definitely interesting to follow this path and add another feature extraction method or try more classifiers. It also has great training and inference speed regarding the fact that it utilizes only CPU unit.

The EfficientNet delivers almost optimal accuracy and we will use it further in this work. The interesting fact is that the model works pretty good for Dataset31. Also we expected augmentation to help with problem of low number of training samples, but it does not seem to have that effect.

4. Rim Size Estimation

This chapter is focused on estimation of rim size. First, we introduce necessary background for this task. Then we describe considered methods and choose one of them, for which we later prepare the data and perform experiments.

4.1 Background

We start this section once again by looking at the properties of our data, mostly in terms of sizes and angles of the objects. Then we describe technical properties and assumptions about rim sizes and in the end we present an approach to measure them.

The first note about data properties is about the cameras. Our cameras were not specially calibrated and they have slight difference in tilt and also in position in which they are mounted.

Furthermore, we have no guarantee about the exact position of the car on the conveyor belt. We only know that its somewhere in the middle of the belt, which can mean differences in up to lower tens of centimeters (rough estimate from the data). Another thing about the car wheels is that we cannot expect the wheel to be exactly perpendicular to the camera because there can be a slight rotation in the car position on the belt or the wheels can be slightly turned to one side. All those properties mean that we cannot base our approach on the specific distances from the camera to the objects or expect objects in specific angles.

In terms of the rim sizes, we do not we do not have the true rim sizes available in our data. As far as we know [45], Škoda Auto rims have diameters in inches in integer values, usually from 15" up to 20". This diameter defines the diameter of the area that is directly under the tire. Unfortunately, this is not the same diameter that we get by naively measuring the rim from the outside. See Figure 4.1. The cause is a part of the rim called fringe. Its purpose is to help to keep the tire in place. The outer diameter of the rim is then the rim diameter plus fringe on both sides.

The fringes can differ in shapes and sizes [46] and it is described in rim specification. Taking into consideration the lack of data about rim diameters and properties of the fringe, we decided not to follow the idea to determine the exact rim diameter and we focus our attention more on the relative comparison of sizes of rims on a single car.

To have at least some estimation of the rim diameter, we can use a different kind of object in the image whose exact size we know. Luckily, one of these objects lies in the center of the wheel and it is the pitch circle defined by the wheel bolts. Its diameter was already mentioned in the first chapter as PCD. To our best knowledge [45], all new Škoda Auto cars, especially models Octavia and Karoq that we have in our data, have wheels mounter by five bolts with pitch circle diameter of 112 mm. This knowledge is used as a base for our methods.

Lastly, we expected that the circumference of the rim in our data would be a nearly perfect circle. This expectation holds when the rim is in the middle of the frame, but as it approaches on of the sides, it is skewed into ellipse. See visualization in Figure 4.2.

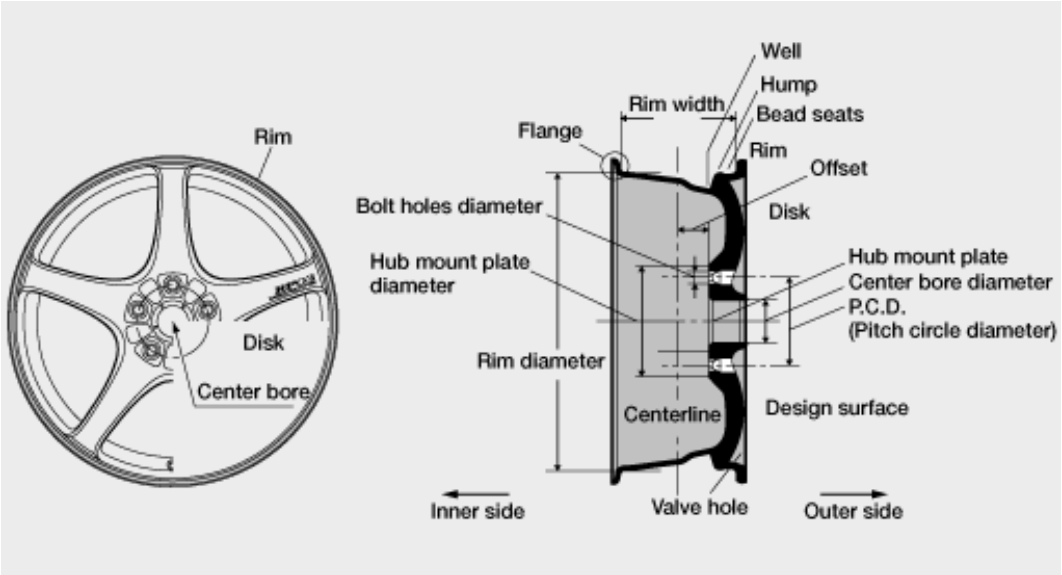


Figure 4.1: Cross section of the rim showing the flange - see the specification of the rim diameter and the flange. (Image taken from ae101.tappsville.com website [12])

Figure 4.2: Frame showing elliptical shape of the rim near the edge of the frame - the green circle is detection of the rim by Hough transform for circles.

4.2 Considered Methods

The approach described above leads to two separate challenges that need to be solved. To compute the pitch circle diameter, the wheel bolts need to be identified. To compute the diameter of the rim, we need to identify the contour of the rim.

For identification of the wheel bolts, we can refer to the second chapter, where segmentation and detection methods are presented. From the blob of pixels of a bolt or its bounding box, we can compute its geometric center and use it for estimation of PCD.

To identify contour of the rim, we can also use segmentation techniques. If we had reliable segmentation method for the rim, the contour extraction would be straightforward. However this does not apply for the bounding box detection techniques, as the bounding box would not define the contour of the rim.

Due to the time requirements of a proper development of the segmentation pipeline and labeling of data, we move the segmentation approach to future work and focus on more simple approach that combines multiple techniques.

4.2.1 Selected approach

First we train detection neural network to detect bolts and the rim. We use YOLOv5 (specifically YOLOv5s model) as we already have training and inference scripts prepared from the second chapter.

By detecting bolts, we can perform the procedure described previously, where we take centers of those bounding boxes and use them for PCD estimation. Because the rim can get ellipse shape near the edges, the same should be expected to happen to the pitch circle. To avoid errors that would arise by forcing circular shape on it, we fit the bolts as an ellipse. To perform the ellipse fit, we use algorithm proposed by Halir and Flusser [18] and implemented in scikit-learn library. The result of this procedure is shown in Figure 4.3a.

The detection of bounding box of the rim can help us remove parts of the image that are not relevant for the rim contour extraction. We simply crop the image to the rectangle defined by the bounding box, which leaves us with image of a rim and small parts of the tire. See image of this type of crop in Figure 4.3b.

To extract contour, we choose to threshold the image and extract the contour. To obtain the thresholded image, the crop is converted to grayscale, blurred to smooth the edges and then converted to binary image based on Otsu's method [29]. The result of thresholding is presented in Figure 4.3c.

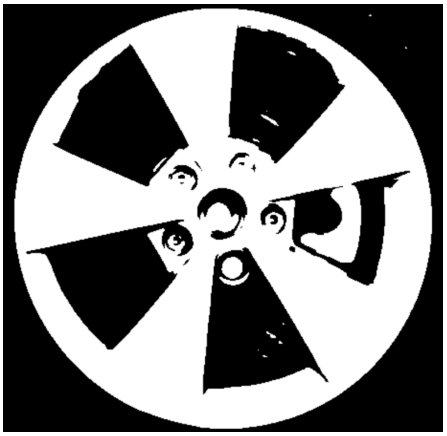
Unfortunately, the contour extraction is not a simple procedure because some rims, especially the dark ones, do not generate stable contour on the circumference of the rim. See example in Figure 4.3d. The contours seems to be most stable near the middle of the edges. This lead us to the custom procedure of contour estimation.

To obtain the most stable sample points of the contour, we cast five vertical or horizontal rays from the middle of each edge. The spacing between individual rays is set to ten percent of the size of the edge. When the ray hits white pixel we add that location as sample point and stop.

Finally, we use the sampled points to fit an ellipse that we pronounce the contour of the rim. Visualization of the entire procedure using rays is presented

(a) Ellipse fit to the bolts.

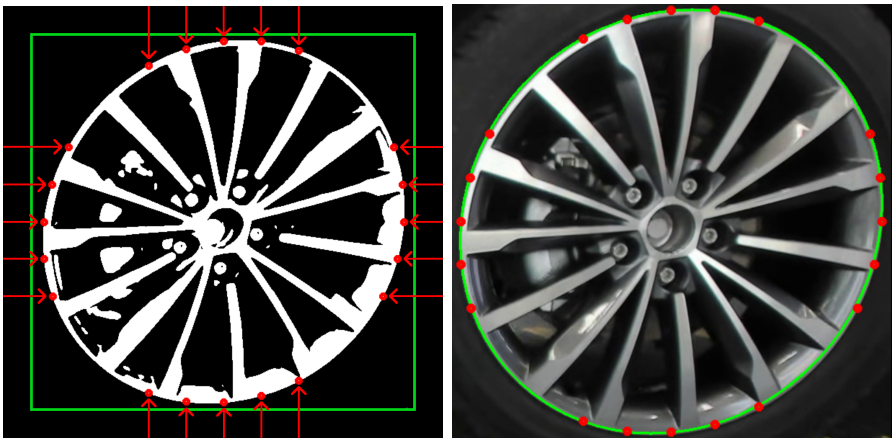
(b) Image cropped to rim box.



(c) Result of Otsu thresholding.

(d) Failure to obtain contour.

Figure 4.3: Pitch circle ellipse fit and stages of rim thresholding.



(a) Visualization of sampling. (b) Ellipse fit

Figure 4.4: Visualization of rays sampling rim contour - the green rectangle is the rim bounding box that defines the image on which we perform the contour extraction procedure.

in Figure 4.4.

The last challenge lies in comparison of ellipses representing pitch circle and rim contour.

We can make following observation. Assume we have a circle in a three-dimensional space. By looking at it from a position perpendicular to the plain the circle lies in, we see its projection as a circle. After an arbitrary rotation of the circle around its center, we will see it as an ellipse (except for edge case where it becomes a single-dimensional shape). The important property of the ellipse is that its major axis size will be equal to the diameter of the circle.

With this knowledge, we can now imagine two circles of different sizes in the same plane in three-dimensional space that share the center. The smaller one will represent pitch circle and the larger one rim contour. Based on the previous observation, by rotating those two circles together, we should always get axis aligned ellipses (both major axes lie on a single line, the same happens for minor axes) and the ratio among the major axes is equal to the ratio of the diameters among the circles.

In simple terms, we should get almost correct ratio between PCD and outer rim diameter by computing ratio between major axes of the ellipses we have extracted. This is a slight simplification of the real scenario because the outer rim contour is not in the same plane as the wheel bolts. The wheel bolts are usually deeper in the wheel.

4.3 Data Preparation

As the base for the data preparation in this chapter, we use Dataset31 from the previous chapter. This dataset already contains crops of the wheels with samples for all the classes. Moreover, the data are split into non-overlapping training, validation and test sets.

We generate 400 training samples by applying random selection from the whole training set of Dataset31. We obtain 100 validation and 100 test samples

by applying the same methodology on their respective sets from Dataset31.

We labeled the wheel bolts and the entire rims with bounding boxes. The labelling was done using CVAT tool the same way as in the second chapter.

4.4 Experiments

As a first experiment the YOLO detection network is trained for wheel bolts and rim. The second experiment is focused on the accuracy of capabilities of our custom contour extraction method.

4.4.1 Bolts and Rim Detection

Setup

The selected model is YOLOv5s. We run the training with default parameters selected by the authors of the model. The details are described in attachment A.1. The training was set for 100 epochs.

Results

The training took 10 minutes. The results of individual classes are in Table 4.1.

Class	**Labels**	**Precision**	**Recall**	**mAP@.5**	**mAP@.5:.95**
Bolt	475	1	0.998	0.995	0.651
Rim	95	0.984	1	0.995	0.993
All	570	0.992	0.999	0.995	0.822

Table 4.1: Results of YOLOv5s for rim and bolts detection on validation data

Both rims and bolts were detected in almost all cases. The metric mAP@.5:.95 for the bolts class shows that the network has problems with predicting the bounding box precisely. This could potentially introduce errors that will propagate to the pitch circle ellipse fit and rim size estimation.

4.4.2 Rim Contour Extraction

Setup

To test our algorithm for contour extraction, we take the validation data of the dataset and see its behavior on the samples where rims have their whole circumference in the image.

Results

Out of 100 validation images, 69 images have the rim circumference fully visible. The 63 rims have a good fit of ellipse to the contour. The remaining 6 predictions were faulty. All of them were of a darker color, which means that the thresholding algorithms did not separate rim and the tire well. This resulted in rays sampling different points than the contour which disrupted the correct fit of the ellipse. See examples of good and bad fit in Figure 4.5.

(a) Example of good ellipse fit.

(b) Example of good ellipse fit.

(c) Example of bad ellipse fit.

(d) Example of bad ellipse fit.

Figure 4.5: Examples of good and bad fit of rim contour ellipse.

4.4.3 Discussion

The detection of the rims works well, but in case of wheel bolts we can see problems with prediction of the correct location. The rim contour extraction algorithm performs good on brighter rims but makes serious mistakes on the darker ones.

The topic estimation of rim diameter by comparison of both ellipses is left for the prototype chapter, where we can observe the values on multiple frames in the row.

5. Prototype

This chapter combines results of the previous chapters to produce a working prototype of a system that tracks rims and checks, whether all rims on a single car are of the same type and size. First we shortly discuss object tracking and introduce a simple tracking heuristic that we use. Then we show, how individual models are connected in order to work as a system. At last, we evaluate its performance.

5.1 Object Tracking

The end objective of the thesis is to verify, whether all rims on a car match. To do that, we need to be able to keep track of individual instances of cars as they move through the scene, as well as track individual wheels and assign them to a specific car. We refer to this task as object tracking. Its goal is to track individual objects as they move around in the video.

To illustrate the challenges that the object tracking algorithm has to face, we will describe the four step methodology that is used in SORT algorithm [6]. Both SORT and its successor DeepSORT [50] are popular algorithms used in object tracking.

The first challenge is object detection. It is described in detail in the second chapter.

The next step is estimation. It models the motion of the object and together with its current position predicts its location in the subsequent frame(s). The commonly used model for modelling motion is Kalman filter [24]. The benefit of modelling motion is that in case of a short occlusion, we have a good estimate of the tracked object's location, so there is chance for the tracking recovery.

The third challenge is data association. When we arrive at a frame, we have estimations of object locations from previous frame and object detections from the current frame. To update object locations for the current frame, we can for example take the estimation and check overlap (intersection over union) with the closest detection. If it is higher than some threshold, we update the motion model and the location of the object.

This can have two problems. The first one is that if we do this approach greedily, some estimates of objects will end up not having feasible detection near them. This can be improved by assigning estimates to detections in order to maximize sum of overlaps. The second problem is identity switch, for example when two running dogs cross paths and one dog occludes the other, a tracking algorithm can accidentally swap their ids, which ruins their tracking further on. This can be improved by using a better metric for estimation and detection assignment. For example we can take into consideration similarity of object features between object in the past and in the current frame.

The last step is to solve object creation and destruction, which means defining conditions, when an unassigned detection becomes an object and when to stop tracking of an object.

5.1.1 Specifics of Our Data

The object tracking is usually presented on a single camera running at fps rate that resemble continuous movement of objects. Our data run at 1 frame per second, which is low, but we have large objects in the frame, so it is still sufficient for data assignment via intersection over union.

Another property of our data are occlusion caused by working staff. It is usually only for one or two frames, but there are also cases, when the wheel is not visible in the scene at all as it passes by.

Because we work with 2 cameras running in parallel recording the same car, we need to maintain a representation of a car present in both cameras. Furthermore we need to deal with slight difference in size and position of the same car in individual cameras because they are not mounted in exactly opposite spots and one of them is slightly tilted. These differences are displayed in Figure 5.1.

Figure 5.1: Custom tracking algorithm - see difference in position and size of a car and its wheel. The camera A in the top and camera B in the bottom.

5.1.2 Implementation

Because we have a system with two cameras and an already prepared object detection model, we have decided to implement a simple tracking algorithm ourselves. The already existing algorithms would have higher accuracy, but would also come with higher cost in implementation or integration time, so we leave them for the future work.

Our tracking approach has two preprocessing steps. The first one is that when two boxes of the same class overlap, we keep only the bigger one. From the nature of our data, we can be sure that this approach is correct and will rule out duplicate detections of the same object. We implement this feature because we have observed case when YOLO generated two bounding boxes for a single wheel. The next preprocessing step is to ignore bounding boxes, that are too close to the left or right edge of the image. For example near the left edge, we ignore all bounding boxes, whose right side is lower than 15 percent of the image. This should help us with inconsistent detections near the edges, so we take into consideration objects only when they are recognizable.

The car tracking is performed only on camera A and this information is used also for camera B. This is acceptable because the differences in horizontal coordinates of a car are not big. The car tracking is implemented in a simple way. We always track only a single car, the data association is done by intersection over union of bounding box in a current and a previous frame. If we lose tracking of a car, we wait for a few frames and then cancel the tracking. When no car is being tracked and bounding boxes appear in the frame once again, we take the leftmost car bounding box and instantiate it as a newly tracked car.

The wheels are tracked in the same way as the car, the only difference is the class of the bounding boxes that we are focused at. The tracked wheel is assigned to the currently tracked car.

For the visual output of our tracking algorithm, see Figure 5.1.

5.2 Prototype structure

The prototype structure and the flow of the data almost completely follows the order in which they were introduced in the thesis. See Figure 5.2 for schema of the most important modules and data they pass to each other.

At first new frames from the cameras arrive. Let us denote frame_a the frame from camera A and frame_b the frame from camera B. The frame_b is flipped horizontally due to the properties of the tracking algorithm and it also looks more coherent in the user interface. In the following modules, we process the frame_a and frame_b the same way and the difference is again important only for the tracking algorithm in the end.

The journey of the frame begins with the object detection model from the second chapter. The model detects the car bounding box that is sent directly to the tracking algorithm and the wheel bounding box that is sent for further processing.

By cropping the frame to the rectangle defined by bounding box, we get an image of the wheel. The wheel image is first passed to the rim classifier from the third chapter that delivers the predicted class to the tracking algorithm.

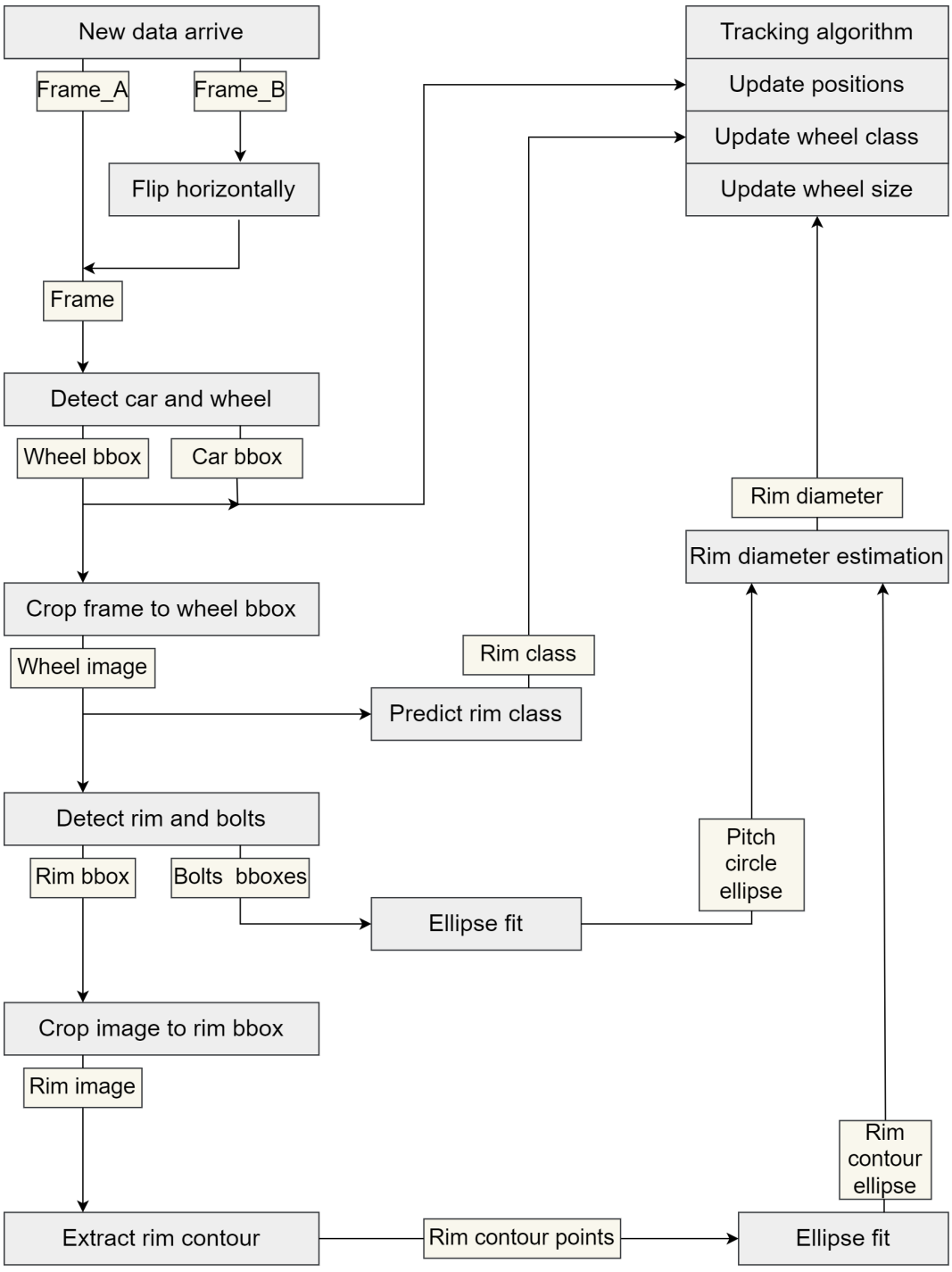


Figure 5.2: Schema of the data flow through the main prototype modules. The modules have gray background. The word 'bbox' is an abbreviation for bounding box.

The wheel image is also sent to the rim and bolts detection model developed in the fourth chapter. The detected bolts bounding boxes go to the ellipse fit algorithm that provides us with the estimate of the pitch circle ellipse. The detected rim bounding box is used for cropping the image further.

The rim image is fed to our rim contour extraction algorithm that extracts samples points from the contour and performs an ellipse fit. The result is estimate of rim contour ellipse. Having both pitch circle ellipse estimate and rim contour ellipse estimate, we can compute the rim diameter and pass it to the tracking algorithm.

The tracking algorithm is the last module of the pipeline. First it updates known position for the car and the wheels. Then it takes into consideration predicted rim class and estimated rim size.

Because we see every wheel in about ten frames, a method to aggregate results of rim class and rim size for a single wheel is needed. For rim class, we decided to do simple voting, where we consider the class of the rim to be the most often predicted class, which seems to be the natural approach. For the wheel size, we aggregate results using an average. The median or weighted average would be also reasonable. For example the weight for the size estimation could be highest when the wheel is near the middle of the frame because we can expect most precise estimations there.

5.3 Results

The results we describe are based on the observations of the prototype behavior on multiple videos from the data. See Figure 5.3 for the prototype user interface we use.

The detection of the car and wheels works as good as can be expected. In rare cases a weird stretch of the car bounding box can be seen when somebody steps in front of the camera and covers large parts of the frame. The wheel classification did not make any mistakes during our observation. Again, in rare cases we have seen that the number of votes for the class was lower than the maximum possible amount, but it was not enough to damage the overall class prediction.

On the other hand the performance of the tracking algorithm and size estimation was not optimal. The tracking algorithm works well in clear environment, but when a person occludes the car in a specific place for more than a few frames, the algorithm can prematurely end tracking of the car and start tracking the same car as a new one.

The size estimation also has issues that were already mentioned in Chapter 4. It is poor estimation of ellipses when the rim is dark that leads to nonsensical estimates of the rim diameter. Estimation on the rims with bright surface performs much better. If we look at the averages of size estimates on individual rims on a single car, they seem to be quite close to each other. We often see that the difference between the average size among the four wheels is less than 0.3 inches, which is surprisingly good considering the errors introduced in bolts detection and other disruptive influences.

The average processing time of the whole pipeline for a single input (two frames, one from each camera) when it is under load, meaning there is car and wheels in both frames, is about 0.4 seconds. When the scene is empty, it is

Figure 5.3: User interface of the prototype - on the left side there is Frame_a and Frame_b, next to them are visualizations of ellipse estimations and predicted classes. The texts under heading 'Camera A/B' sum up information about size estimation for the current frame. The texts under headings 'Wheel classes/sizes' sum up aggregated knowledge we have about the classes and sizes of the rims on the currently tracked car.

about 0.15 seconds. This is very promising because in spite of that we have done almost no speed or memory optimizations, the prototype is still able to process the inputs in real-time.

Conclusion

The goal of the thesis was to make a prototype of a real-time system that is able to validate whether all car rims match in type and size.

The first chapter introduced the topic of car manufacture and presented the data screening conditions. Furthermore were described the challenging properties of the data.

The second chapter was focused on the detection of a car and wheels. We mentioned related work, prepared the data and trained two fairly successful models: Hough transform and YOLOv5 detection network.

The third chapter was dedicated to the task of wheel classification based on the differences in shape and color. We prepared the data and tested two classification methods: a combination of HOG features and SVM classifier, and EfficientNet classification network. The former did not reach practical results, but had a potential to be improved. The latter reached almost optimal accuracy when it was supplied with enough data for each class.

The fourth chapter discussed facts we knew about the rim size and we have proposed a custom procedure that is able to estimate pitch circle and rim contour. Our procedure worked quite reliably with brighter rims and failed in case of the darker ones. This problem could have been solved by introducing segmentation neural network instead of simple thresholding. The comparison of sizes of brighter rims on a single car seemed to give stable results, but for a practical use it needs tuned to be tuned more accurately.

The fifth chapter used previously developed methods to build a functional prototype. First, we took a step aside and introduced a simple custom heuristic for car and wheel tracking. Then the models were connected into a single pipeline that detects wheels, measures and classifies them, and also sums up these pieces of information for the whole car. Although tracking heuristic and rim estimation needs to be improved for practical use, the final system works really well, processes data in real-time and meets all the objectives that were specified in the beginning.

During the making of the thesis, we have collected multiple ideas for the future work that would improve both the data preparation and the final system.

5.4 Future work

Unknown rim class - premise of this thesis is that users of the system are able to obtain images of all classes of rims that the system can encounter. This means that the system should never see a class of a rim that is missing in the training data. In case this would happen, the system behavior would not be well defined and it would likely predict the most similar class.

The system can be improved to recognize an unknown class. This task is called open set recognition. A simple way to implement it would be a threshold on probability of a predicted class that would distinguish between confident prediction and unknown class. An inspiration for a more accurate method can be work of Bendale and Boult [5] which proposes a new layer of neural network that support open set recognition, or survey by Geng et al. [14] focused on the topic of open set recognition as a whole.

Automatization of labeling - since labeling of data can be a time and attention demanding task, we suggest to use semi-automatic approaches for it, where user's action are used more for control of the correctness of the label rather than its creation. One example is object detection in CVAT tool. User can either use integrated YOLOv4 network to label the image and then modify labels, or use models that can predict contour of a selected object.

Second example would be in classification. One idea is that user would select a representative picture of a class and an algorithm would extract similar images from a dataset. Second idea is an algorithm that would partition dataset into several groups based on similarity by itself. We have performed a small experiment during the development of this thesis where image features were represented by the latent space of an autoencoder [3]. Images were then clustered into several groups using Gaussian mixture model. This approach yielded some results, but they were not good enough for practical use. A possible improvement may use different features, different autoencoder such as disentangled autoencoder [3], different clustering algorithm or different pipeline at all.

Improvement of tracking - there is a lot of room for improvement because we have used a custom simple algorithm. Ideally, a widely respected algorithm such as DeepSORT should be integrated instead. Furthermore, one needs to think through what to do in cases when the system arrives into an inconsistent state. For example the system could detect only three wheels, or there could be an occlusion and it could split one real wheel into two tracked entities, which would end up in three detected wheels on one side of the car, etc.

Bibliography

[1] Ma'moun Al-Smadi, Khairi Abdulrahim, and Rosalina Abdul Salam. Traffic surveillance: A review of vision based vehicle detection, recognition and tracking, 02 2016.

[2] Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-dujaili, Ye Duan, Omran Al-Shamma, Jesus Santamaría, Mohammed Abdulraheem Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. *Journal of Big Data*, 8, 2021.

[3] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *CoRR*, abs/2003.05991, 2020. URL `https://arxiv.org/abs/2003.05991`.

[4] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, pages 404–417, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-33833-8.

[5] Abhijit Bendale and Terrance E. Boult. Towards open set deep networks. *CoRR*, abs/1511.06233, 2015. URL `http://arxiv.org/abs/1511.06233`.

[6] Alex Bewley, ZongYuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. *CoRR*, abs/1602.00763, 2016. URL `http://arxiv.org/abs/1602.00763`.

[7] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934, 2020. URL `https://arxiv.org/abs/2004.10934`.

[8] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[9] A. Buslaev, A. Parinov, E. Khvedchenya, V. I. Iglovikov, and A. A. Kalinin. Albumentations: fast and flexible image augmentations. *ArXiv e-prints*, 2018.

[10] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8:679 – 698, 12 1986. doi: 10.1109/TPAMI.1986.4767851.

[11] Alberto Chávez-Aragón, Robert Laganière, and Pierre Payeur. Vision-based detection and labelling of multiple vehicle parts. In *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1273–1278, 2011. doi: 10.1109/ITSC.2011.6083072.

[12] CRAZY CAR MOTORSPORT. Wheel size, offset, pcd, clearance, tyre size, explained with illustrations. [Online; accessed 3-May-2022]. URL: `http://ae101.tappsville.com/help/wheels.php`, 2008.

[13] Mohamed Elgendy. *Deep Learning for Vision Systems - Chapter 7 - Object detection with R-CNN, SSD, and YOLO*. 1st edition. Manning, 2020. ISBN 9781617296192.

[14] Chuanxing Geng, Sheng-Jun Huang, and Songcan Chen. Recent advances in open set recognition: A survey. *CoRR*, abs/1811.08581, 2018. URL http://arxiv.org/abs/1811.08581.

[15] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. URL http://arxiv.org/abs/1504.08083.

[16] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013. URL http://arxiv.org/abs/1311.2524.

[17] Anton Grigoryev, Dmitry Bocharov, Arseniy Terekhin, and Dmitry Nikolaev. Vision-based vehicle wheel detector and axle counter. *Proceedings - 29th European Conference on Modelling and Simulation, ECMS 2015*, pages 521–526, 05 2015. doi: 10.7148/2015-0521.

[18] Radim Halir and Jan Flusser. Numerically stable direct least squares fitting of ellipses, 1998.

[19] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017. URL http://arxiv.org/abs/1703.06870.

[20] Histogram of Oriented Gradients explained using OpenCV. Histogram of oriented gradients explained using opencv. [Online; accessed 3-May-2022]. URL: https://learnopencv.com/histogram-of-oriented-gradients/, 2016.

[21] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *CoRR*, abs/1709.01507, 2017. URL http://arxiv.org/abs/1709.01507.

[22] Karin Hultström. Image based wheel detection using random forest classification. Master's thesis, Lund university, Faculty of Engineering Centre for Mathematical Sciences Mathematics, 2013. URL https://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=3457767&fileOId=3459875.

[23] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, TaoXie, Jiacong Fang, imyhxy, Kalen Michael, Lorna, Abhiram V, Diego Montes, Jebastin Nadar, Laughing, tkianai, yxNONG, Piotr Skalski, Zhiqiang Wang, Adam Hogan, Cristi Fati, Lorenzo Mammana, AlexWang1900, Deep Patel, Ding Yiwei, Felix You, Jan Hajek, Laurentiu Diaconu, and Mai Thanh Minh. ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference, February 2022. URL https://doi.org/10.5281/zenodo.6222936.

[24] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[25] Michael Kass, Andrew P. Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1:321–331, 2004.

[26] Yann Lecun, Leon Bottou, Y. Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278 – 2324, 12 1998. doi: 10.1109/5.726791.

[27] Günter Leister. *Passenger Car Tires and Wheels*. 1st edition. Springer, Cham, 2018. ISBN 978-3-319-50117-8.

[28] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multi-box detector. *CoRR*, abs/1512.02325, 2015. URL `http://arxiv.org/abs/1512.02325`.

[29] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979. doi: 10.1109/TSMC.1979.4310076.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.

[31] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016. URL `http://arxiv.org/abs/1612.08242`.

[32] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 04 2018.

[33] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. URL `http://arxiv.org/abs/1506.02640`.

[34] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. URL `http://arxiv.org/abs/1506.01497`.

[35] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL `http://arxiv.org/abs/1505.04597`.

[36] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: an efficient alternative to sift or surf, 11 2011.

[37] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3): 211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[38] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018. URL http://arxiv.org/abs/1801.04381.

[39] Boris Sekachev, Nikita Manovich, Maxim Zhiltsov, Andrey Zhavoronkov, Dmitry Kalinin, Ben Hoff, TOsmanov, Dmitry Kruchinin, Artyom Zankevich, DmitriySidnev, Maksim Markelov, Johannes222, Mathis Chenuet, a andre, telenachos, Aleksandr Melnikov, Jijoong Kim, Liron Ilouz, Nikita Glazov, Priya4607, Rush Tehrani, Seungwon Jeong, Vladimir Skubriev, Sebastian Yonekura, vugia truong, zliang7, lizhming, and Tritin Truong. opencv/cvat: v1.1.0, August 2020. URL https://doi.org/10.5281/zenodo.4009388.

[40] Allam Shehata, Sherien Mohammad, Mohamed Abdallah, and Mohammad Ragab. A survey on hough transform, theory, techniques and applications, 02 2015.

[41] Simple Background Estimation in Videos using OpenCV (C++/Python). Simple background estimation in videos using opencv (c++/python). [Online; accessed 16-November-2021]. URL: https://learnopencv.com/simple-background-estimation-in-videos-using-opencv-c-python/, 8 2019.

[42] Chandan Singh and Nitin Bhatia. A fast decision technique for hierarchical hough transform for line detection, 07 2010.

[43] Huansheng Song, Haoxiang Liang, Huaiyu Li, Zhe Dai, and Xu Yun. Vision-based vehicle detection and counting system using deep learning in highway scenes. *European Transport Research Review*, 11, 12 2019. doi: 10.1186/s12544-019-0390-4.

[44] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019. URL http://arxiv.org/abs/1905.11946.

[45] The world's largest wheel fitment database. The world's largest wheel fitment database. [Online; accessed 3-May-2022]. URL: https://www.wheel-size.com/size/skoda/, 2022.

[46] Tyre Size Calculator. Wheel/rim profiles of passenger cars. [Online; accessed 3-May-2022]. URL: https://www.tyresizecalculator.com/wheels/wheel-rim-profiles, 2022.

[47] Veerapathirapillai Vinoharan, Amirthalingam Ramanan, and Saluka Kodituwakku. A wheel-based side-view car detection using snake algorithm, 09 2012.

[48] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features, 02 2001.

[49] Dominik Vít. Automated visual inspection system for a car engine space. Master's thesis, Czech technical university in Prague, Faculty of Nuclear Sciences and Physical Engineering, 2020. URL <http://adamnovozamsky.com/files/2020_Diploma_Vit.pdf>.

[50] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. *CoRR*, abs/1703.07402, 2017. URL http://arxiv.org/abs/1703.07402.

[51] HK. Yuen, J. Princen, J. Illingworth, and J. Kittler. Comparative study of hough transform methods for circle finding. *Image and Vision Computing*, 8(1):71–77, 1990. ISSN 0262-8856. doi: https://doi.org/10.1016/0262-8856(90)90059-E. URL https://www.sciencedirect.com/science/article/pii/026288569090059E.

[52] Škoda Auto storyboard website. Škoda octavia – záběry z výroby. [Online; accessed 8-November-2021]. URL: https://www.skoda-storyboard.com/cs/170720-skoda-octavia-production-footage-2/, 8 2017.

[53] Škoda Auto storyboard website. Škoda auto produced more than 750,000 vehicles at its czech plants in 2020 despite covid-19 pandemic. [Online; accessed 8-November-2021]. URL: https://www.skoda-storyboard.com/en/press-releases/skoda-auto-produced-more-than-750000-vehicles-at-its-czech-plants-in-2020-despite-covid-19-pandemic/, 1 2021.

[54] Škoda Auto website. Octavia — Škoda auto a.s. [Online; accessed 9-November-2021]. URL: https://www.skoda-auto.cz/modely/octavia/octavia, 2021.

List of Figures

List of Tables

List of Abbreviations

1. **PCD** - Pitch Circle Diameter
2. **CVAT** - Computer Vision Annotation Tool
3. **HOG** - Histogram of Gradients
4. **SVM** - Support Vector Machines

A. Attachment

A.1 Electronic Attachment

The electronic attachment contains code and data used in the thesis. The documentation files that further describe the structure and use are:

1. User Documentation
   (a) Introduces the usage of the prototype.
   (b) Located in `./docs/User_documentation.pdf`
2. Technical Documentation
   (a) Describes the whole project structure, installation and data processing scripts.
   (b) Located in `./docs/Technical_documentation.pdf`
3. Experiments Documentation
   (a) Points both to scripts that were used for experiments and the data available for individual runs presented in the thesis.
   (b) Located in `./docs/Experiments_documentation.pdf`