technische universität
dortmund

Masterarbeit

Development of a perceptible prototype for detection and classification of food inside refrigerators

Christopher Kukkel
Februar 2020

Gutachter:
Prof. Dr. Jian-Jia Chen
Dr.-Ing. Christoph Krimpmann

Technische Universität Dortmund
Fakultät für Informatik
Design Automation of Embedded Systems Group
http://ls12-www.cs.tu-dortmund.de

In Kooperation mit:
Smart Mechatronics GmbH

Abstract

Applications in the field of smart home are now finding their way into many households and enable a consumer to automate and simplify various tasks. In this thesis a prototype is developed, which is able to detect food in a refrigerator and to classify it. This allows a consumer to know at any time, what is in his refrigerator and certain food products could be automatically re-ordered. A main goal of this thesis is to enable the detection and evaluation of the refrigerator contents to be performed on an embedded system. The selection of objects is limited to the following five classes: bananas, beer bottles, broccoli, cucumbers and tomatoes. For evaluation purposes, different data sets are created, which show these objects in different constellations. First, different object detectors are compared. Faster R-CNN Inception V2 has achieved the best results. Then several methods are tested to improve the object detector even further. For this purpose, the aspect ratios of the anchor boxes are modified and the confidence threshold for detection is lowered. This way, the mAP can be increased by 14.8 % and reaches a value of 56.26 %. To further improve the detection of newly added food, a pipeline is developed that combines Faster R-CNN with background subtraction. This pipeline is ported on a Raspberry Pi 4 and reaches a run time of 40 seconds.

Acknowledgements

I would like to thank Prof. Dr. Jian-Jia Chen for the opportunity to work on my own topic in the form of a master thesis and for the helpful support I received.
I would also like to express my special gratitude to Dr.-Ing. Christoph Krimpmann and Christian Dopp, who were always available for me and supported me with many valuable ideas and advice.

Contents

1 Introduction **1**

1.1 Motivation . . . 1

1.2 Goal . . . 1

1.3 Structure of the thesis . . . 2

2 Neural Networks **3**

2.1 Convolutional neural networks . . . 4

2.1.1 Convolutional layer . . . 4

2.1.2 Detector stage . . . 5

2.1.3 Pooling layer . . . 6

3 Object detection **9**

3.1 Overview . . . 9

3.2 History of Computer Vision . . . 9

3.2.1 Feature detection . . . 10

3.3 Watershed for image segmentation . . . 12

4 Object detection using Neural Networks **15**

4.1 Feature detectors . . . 15

4.1.1 VGG-16 . . . 15

4.1.2 Inception V2 . . . 16

4.1.3 MobileNet . . . 17

4.2 Region based convolutional neural networks . . . 19

4.3 Region based fully convolutional network (R-FCN) . . . 24

4.4 Single Shot Multibox Detector (SSD) . . . 24

4.5 You Only Look Once (YOLO) . . . 25

4.6 Transfer learning . . . 26

5 Implementation and data set **27**

5.1 Used frameworks . . . 27

5.2 Data sets . . . 28

5.2.1 Evaluation data sets 29
5.3 Prototype 32

6 Metrics used in the evaluation 35
6.1 Mean Average Precision 35
6.2 Custom metric 36

7 Evaluation of the detection results 39
7.1 Literature review 39
7.2 Training on evaluation data set 41
7.3 Results using the first data set 41
7.4 Evaluation on the second data set 44
7.5 Evaluation on the third data set 45
7.6 Background subtraction 56
7.7 Evaluation of the detection speed 60

8 Discussion 63

9 Conclusion & Outlook 67

Appendices 69

List of Figures 76

Bibliography 82

Eidesstattliche Versicherung 83

Acronyms

AP	*Average Precision*
ANN	*Artificial Neural Network*
BRIEF	*Binary Robust Independent Elementary Features*
CNN	*Convolutional Neural Network*
CUDA	Compute Unified Device Architecture
DoG	*Difference of Gaussian*
FAST	*Features From Accelerated Segment Test*
Faster R-CNN	*Faster Region Based Neural Network*
FN	*False Negative*
FP	*False Positive*
ILSVRC	*ImageNet Large Scale Visual Recognition Challenge*
IoU	*Intersection over Union*
mAP	*Mean Average Precision*
MPN	*McCulloch-and-Pitts-Neuron*
NAS	*Neural Architecture Search*
ORB	*Oriented FAST and rotated BRIEF*
ReLU	*Rectified Linear Unit*
R-FCN	*Region-based Fully Convolutional Network*
ROI	*Region of Interest*
RPN	*Region Proposal Network*
SCERPO	*Spatial Correspondence, Evidential Reasoning and Perceptual Organization*
SIFT	*Scale Invariant Feature Transform*
SSD	*Single Shot Multibox Detector*
SURF	*Speeded Up Robust Features*

TP *True Positive*

YOLOv3 *You Only Look Once (Version 3)*

Chapter 1

Introduction

This thesis shows the development process of a prototype, which is able to detect food inside a refrigerator.

1.1 Motivation

German households produce the equivalent of around 85 kilograms of food waste per inhabitant per year. From this amount 37 kilograms are theoretically avoidable [48]. This means that more than 3 billion tons of food are thrown away throughout Germany, which theoretically could have been avoided. One way to reduce this waste would be automatic monitoring of the groceries purchased. This would make it possible to record which and how much food is still in stock, when it was bought and how many days it will remain fresh.
Another advantage of monitoring food in a household is a feature, that make everyday life easier. For example, a user of food monitoring could see when shopping what items are already in the household and what items might need to be bought. A user could also receive suggestions for recipes containing food that is already in the home. This information could even be used to automatically reorder certain foods.

1.2 Goal

The goal of this master thesis is to develop a prototype for food recognition inside a refrigerator. The focus here is on the object detection, which is to be accomplished by using a camera. In the process of the master thesis, different machine learning models for object recognition should be compared for this application. Subsequently, it will be examined how this object recognition can be further improved for this application case. It should be noted that this object recognition should be performed on an embedded system, such as a Raspberry Pi, which is a general purpose single-board computer. The final

prototype shall then be able to take images, evaluate them and make the information about the detected food available to a user, for example via a web interface.

1.3 Structure of the thesis

This thesis is divided into the following chapters: In chapter 2 the basics of neural networks are explained. In the following chapter 3, different methods for object and feature detection are described. Various approaches for object detection using neural networks are presented in chapter 4. The chapter 5 describes the frameworks used in this thesis, the data sets used and technical details about the prototype. The metrics, which are used in evaluation, are presented in chapter 6. In chapter 7 different methods for object recognition with the help of the created data sets are evaluated. Subsequently, several approaches are tested to further improve the performance of the object detectors and the detection of objects in a refrigerator. In conclusion, the results are discussed in chapter 8 and a summary and outlook on the following work is given in chapter 9.

Chapter 2

Neural Networks

Modern methods to recognize objects in images or extract features use neural networks. The foundation for this was laid in 1943 with the development of the *Artificial Neural Network* (ANN) by McCulloch and Pitts [36]. The authors try to describe and model the behavior of human brain cells by means of a mathematical model. The presented *McCulloch-and-Pitts-Neuron* (MPN) receives several values as input, which are aggregated by a function g. The function f is then used to check whether a certain threshold value is exceeded. Depending on this, a certain output value is then returned. A graphical representation can be found in Figure 2.1.

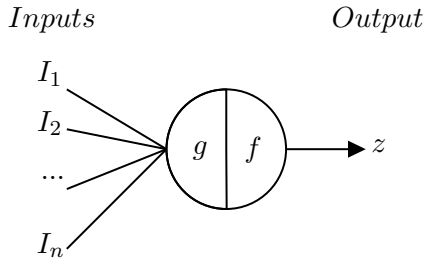


Figure 2.1: Graphical representation of an *McCulloch-and-Pitts-Neuron*. Adapted from [10].

In 1958 Rosenblatt [43] presented the perceptron, which is a further development of the MPN. Unlike the MPN, each input is given a different weighting. The threshold value is also chosen differently for each perceptron. The output value is normalized to an interval between -1 and 1. To enable the perceptron to learn, all weights are initialized with random values. Next, a subset of the data set is fed into the perceptron and the output is compared to the ground truth. These operations are repeatedly executed till the intended functionality is reached. A visualization of a perceptron is shown in Figure 2.2.

If several of these perceptrons are now connected in series, an ANN is created. To be able to train the weights of the individual perceptrons more efficiently, Rummelhart et al. developed the backpropagation method [45]. The first convolution network was presented by Fukushima in 1979, which is called neocognitron [17]. It is similar to the architecture

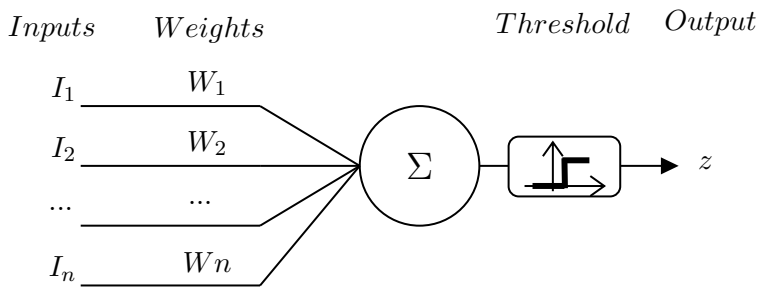


Figure 2.2: Visualization of a perceptron. Adapted from [5].

of modern networks because it uses different stages to extract features, which then are used to perform classification tasks.

One of the first neural networks used to evaluate images was the LeNet, developed by Le Cun et al. in 1990 [11]. With this network it was possible to classify handwritten digits from the MNIST data set. It used a convolution network, followed by a fully connected layer. Backpropagation was also used in this approach.

2.1 Convolutional neural networks

For object detectors or feature extractors, usually *Convolutional Neural Network* (CNN) are used. In the first step, the input image is reduced to a defined size and then fed into a tensor. The dimensions of the tensor correspond to the width and height, as well as the number of color channels of the image. This tensor is then transferred to the first input layer of CNN. This is followed by a sequence of convolutional, detection and pooling stages, called the *feature extraction layer*. At the end of the network, there are one or more fully connected layers that help classify the original image.

2.1.1 Convolutional layer

A convolution layer is usually used to detect certain features in an image, such as corners or edges. The computational operation used to do this can be described as a discrete convolution:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a).$$

In this case the input vector is x and w is the kernel. The output tensor s is called the *feature map*. In the case of images, the kernel is often a two-dimensional array. The input is also a two-dimensional array for monochrome images and a tensor for color images, consisting of a two-dimensional array for each color channel. The following formula describes the convolution for a monochrome image I, which exists as a two-dimensional array. The kernel K also has two dimensions:

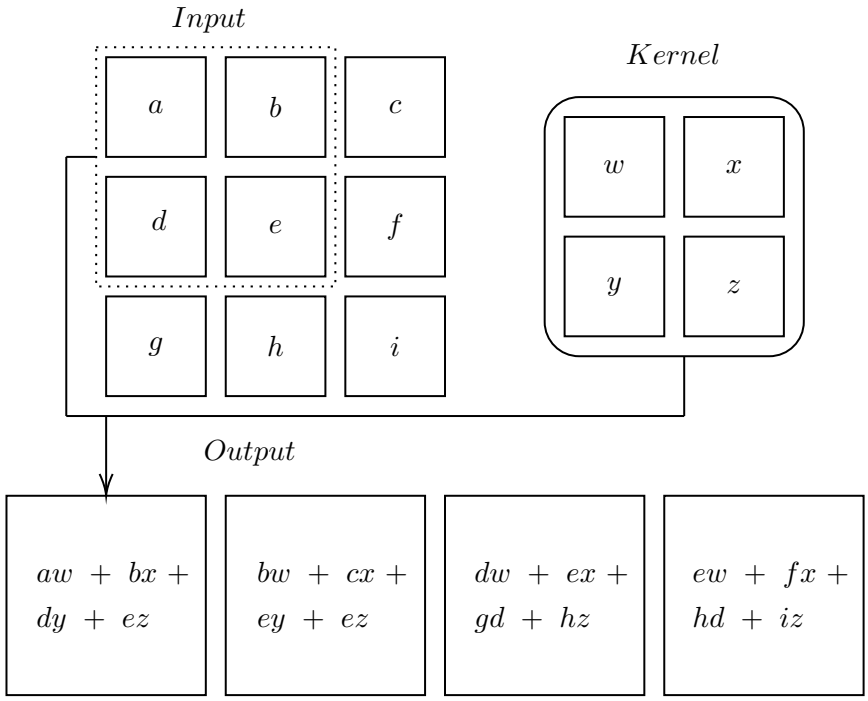


Figure 2.3: Example of a two-dimensional convolution. The kernel is applied to four input values at ones. This creates one output value. Graphics adapted from [20].

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n).$$

A graphical representation of this method can be found in figure 2.3. With the help of this approach it is possible to perform extensive image operations without using complex matrix multiplications. A simple example is the detection of edges in an image. For this operation a convolution kernel with two elements is sufficient. This kernel is used to subtract the value of the left neighboring pixel from each pixel of the output image [20].

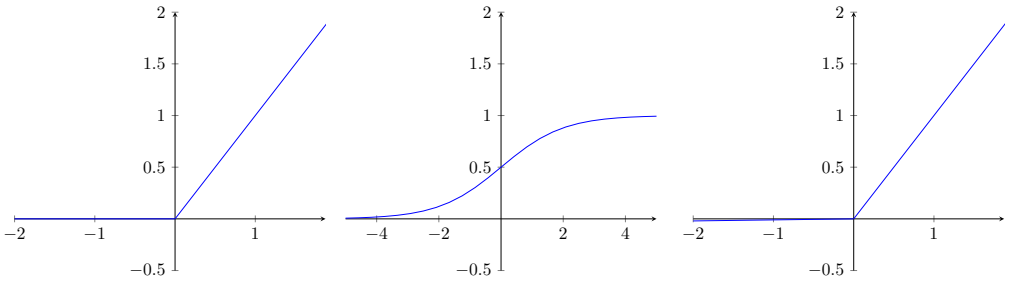
2.1.2 Detector stage

Typically, each of the convolutional layers is followed by a detector layer [20]. Often *Rectified Linear Unit* (ReLU) [21] operators are used as the activation function [52]. The ReLU operator is defined as

$$ReLU(x) = \max(0, x).$$

Other functions that have a sigmoid curve are also conceivable [29], such as

$$Sigmoid(x) = \frac{1}{1 + e^{-x}}.$$



(a) ReLU **(b)** Sigmoid **(c)** Leaky ReLU

Figure 2.4: Examples of the different activation functions.

An advantage of the ReLU function is that the loss of information, even after several layers, is relatively low [37]. A variation of the ReLU function is called *Leaky ReLU* [35]:

$$Leaky\ ReLU(x) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geqq 0 \end{cases}.$$

The advantage of this function is, that it does not output 0 for negative input values. If this would be the case, the following neurons in the next layers receive a 0 as input value. This could let to the non-activation for some neurons in a large portions of the learning process. Plots of the mentioned activation functions can be found in figure 2.4.

2.1.3 Pooling layer

The features found by the convolutional layers range from the simple known features in the first layers to the more complex features in the deeper layers. These features and the position of these features can be found in the feature map. However, small changes in position, such as resizing or rotation of the features, can lead to a different feature map. One approach to avoid this problem is to downsample the data with the help of pooling layers, which reduces the size of the feature map. Known approaches to this are *Average Pooling* [38] and *Maximum Pooling* [58]. *Average Pooling* is the calculation of the average value for a section of the feature map. *Maximum Pooling*, on the other hand, calculates the largest value for an area. Examples for these methods can be found in figure 2.5. The output of the pooling layer is now a reduced version of the feature map, on which smaller changes of the image have less effect [20].

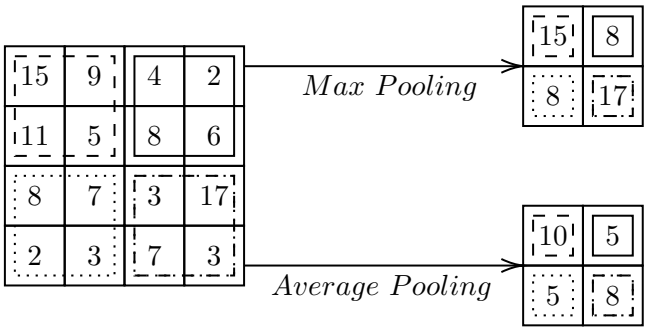


Figure 2.5: Example of *Max Pooling* and *Average Pooling*. Adapted from [38].

Chapter 3

Object detection

This chapter first describes the terms related to object recognition that are used within this thesis. Afterwards, it describes how computer based object recognition has evolved over time. Object detection utilizing neural networks are discussed in chapter 4.

3.1 Overview

To recognize objects in an image, it must first be checked where an object is located (object localization) and subsequently what kind of an object it represents (object classification) [57]. In most approaches, the following two steps are used for this:

Informative region selection: Since objects can appear in any part of the image, the first step is to check where a possible object is located. A possible approach is Watershed, which tries to separate objects from the background. Other methods to solve this task, which are using neural networks, are described in chapter 4.

Feature extraction: In order to recognize objects, certain features must be found by which an object can be identified, even if it is located in a different position or at a different size on the image. Some approaches for feature extraction are describes in subsection 3.2.1.

3.2 History of Computer Vision

First attempts to recognize objects in pictures were made in 1963 by Roberts [42]. Photos were seen as perspective projections to further process images as three-dimensional data. For this reason, linear or perspective transformations can be performed on these images. Lowe [33] presented an approach in 1985 in which three-dimensional objects could be recognized on a picture by two-dimensional features, which was titled *Spatial Correspondence,*

Evidential Reasoning and Perceptual Organization (SCERPO). With this approach, objects are recognized by identifying locally corresponding features in an image.
Another approach was introduced by Huttenlocher and Ullmann in 1987 [27]. The authors try to transform points from an three-dimensional model coordinate frame to a two-dimensional image coordinate frame. This transformation is used to predict feasible alignments of a image with a model.

The method for object recognition presented by Zisserman et al. [60] in 1995 is based on first separating an object into individual simple geometric objects and assigning them to a class, such as rotational surfaces, pipes or polyhedra.

3.2.1 Feature detection

In order to create specific connections between two images, the recognition of certain features is necessary. A feature is a specific area in an image, such as edges, corners or blobs. In the field of computer vision, the following requirements are required of features and feature detectors [23]:

Robustness: A feature should also be recognized if the image is re-scaled, rotated, shifted or deformed. In addition, compression artifacts and noise should not affect the recognition of a feature.

Repeatability: A feature should also be recognized under different viewing conditions.

Accuracy: If a feature is searched several times, exactly the same pixels should be recognized as the same feature.

A method to find certain features in images was published by Harris and Stephens [22] in 1988. Here the recognition of corners and edges of an object is used to recognize this object in another image.

Scale Invariant Feature Transformation (SIFT)

A disadvantage of the feature detection methods known so far, as well as the method presented by Harris and Stephens, is that they are not scale invariant. This property is necessary to be able to detect corners in different scales of an image.

In 2004 Lowe introduced the *Scale Invariant Feature Transform* (SIFT) [34]. This approach of finding points of interest utilizes the *Difference of Gaussian* (DoG). The

function $L(x, y, \sigma)$, that represents the scale space of an image, is calculated using the input image $I(x, y)$, which is then convoluted with a Gaussian filter $G(x, y, \sigma)$:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y),$$

where the Gaussian operator is defined as

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}.$$

The image is scaled in different resolutions, where each iteration is half the size of the previous one. The different scales are called octaves and each octave contains several versions of a scaled image. In the next step, a new set of images is created, called the *Difference of Gaussian* (DoG). Each image of an octave is folded through the Gaussian filter, where the σ differs. By subtracting two images, the DoG is calculated.

If the DoG is found, the local extrema are searched within a scale. A pixel is then compared with all its eight neighbors and the corresponding nine pixels on the previous and subsequent image on the octave. This pixel is a possible keypoint, when a local extremum is found.

In order to remove the unusable points from these possible points, the contrast of each pixel is also compared with a threshold value. If the value is below a certain threshold value, this point is no longer included. Furthermore, pixels do not count to the result set, which are located on an edge, but cannot be identified exactly caused by noise. For this purpose the 2×2 Hessian matrix of a pixel is calculated to determine its eigenvalues. If the difference of these values is below a certain threshold value, the observed point is taken as keypoint. The last step is to determine the orientation for each keypoint. This is done by calculating the gradient magnitude and direction for the area around each keypoint.

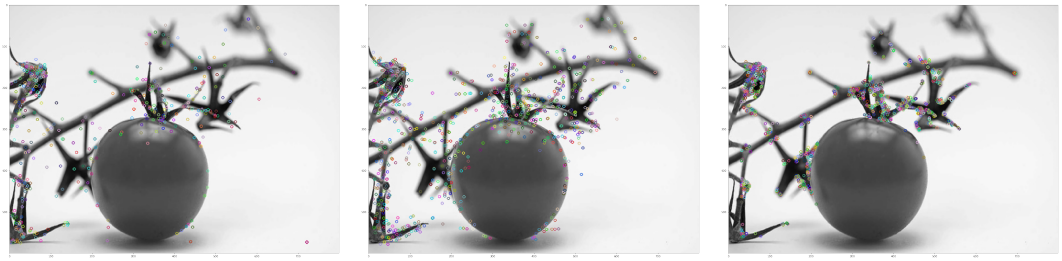
Other methods for feature detection

Another method to identify keypoints in an image, was introduced in 2006 by Bay et. al., called *Speeded Up Robust Features* (SURF) [6]. In this approach the Gaussian filter was replaced by a Box Filter, which generates the *Laplacian of Gaussian*. This leads to an improvement in calculation speed. As alternative to SIFT and SURF Rublee et al. developed *Oriented FAST and rotated BRIEF* (ORB). ORB has similar perforce as SIFT or SURF respectively. However, ORB is not subject to any patent and can therefore be used free of charge. This approach is based on the keypoint detector *Features From Accelerated Segment Test* (FAST) [44] and the descriptor *Binary Robust Independent Elementary Features* (BRIEF)[8].

Figure 3.1 shows an example for each of the described feature detection algorithms. An image of a tomato was used, to show which keypoints are detected by the different

approaches. It is particularly noticeable that the feature points found are on the edge of the tomato or on the background area. This leads to the assumption that the smooth surface of the tomato has little or no features that can be used for identification.
An additional disadvantage of these methods is that the images must first be converted into a gray scale image. This causes most of the color information to be lost, which can also be important for classification.

For this reasons, this approach to detect objects was not pursued further in this thesis and instead an object detection approach was used that uses neural networks.



(a) SIFT **(b)** SURF **(c)** ORB

Figure 3.1: Detected features using SIFT, SURF and ORB. Larger images can be found in the appendix: 9.

3.3 Watershed for image segmentation

The watershed algorithm describes a method to segment images, which was introduced by Beucher and Lantuejoul in 1979 [7]. In this method, the image is first converted into a two-color image. Then the picture is seen as a topographic surface. Areas with a dark shade are considered as lower levels and bright areas as higher levels. The next step is to start filling each of these local minima with different colored water. To prevent the different areas from mixing, barriers will be created at the edge of each local minimum. This process is carried out until the entire area is under water. The barriers created during this process then determine the segmentation of the image. One reason why this method is not applicable to all images is the conversion to a two-color image. An example of this can be found in figure 3.2. Here it is easy to see that, for example, shadows or objects that have a gradient can lead to the contours of an object no longer being visible. This means that some objects cannot be precisely segmented.

(a) Original image.

(b) Image converted into a two-color image.

(c) Segmented image using the watershed algorithm.

Figure 3.2: Example of the watershed algorithm.

Chapter 4

Object detection using Neural Networks

Object detectors, which use neural networks for the recognition of objects, can be divided into two classes.

Two-stage detectors use multiple connected neural networks to find and identify an object on an image. The first of these networks is a feature extractor, which is used to create feature maps. Examples of feature extractors are Inception or MobileNet, which are explained in more detail in the following sections. This feature map is then used in the second stage of the detector. Here areas are determined, where objects could be located. Then, it is checked whether and, if so, which object was found in the previously determined area.

One-stage detectors on the other hand do not determine regions in which objects are suspected. A feature extractor network is followed by a number of convolution layers, which are used to locate an object. Examples of these one-stage object detectors are SSD and YOLO, which are explained in section 4.4 and section 4.5.

4.1 Feature detectors

The detection of certain features in necessary for the training of neural networks to be able to identify unknown objects in images. The requirements for these feature detectors are already discussed in subsection 3.2.1. This section describes some of the feature detectors that were used in the context of this thesis.

4.1.1 VGG-16

A well-known deep neural network used in many object detectors is the VGG-16, which was introduced by Simonyan and Zisserman in 2014 [49]. The architecture is shown in figure 4.1. The input image must first be scaled to a size of 224×224 pixels. Afterwards,

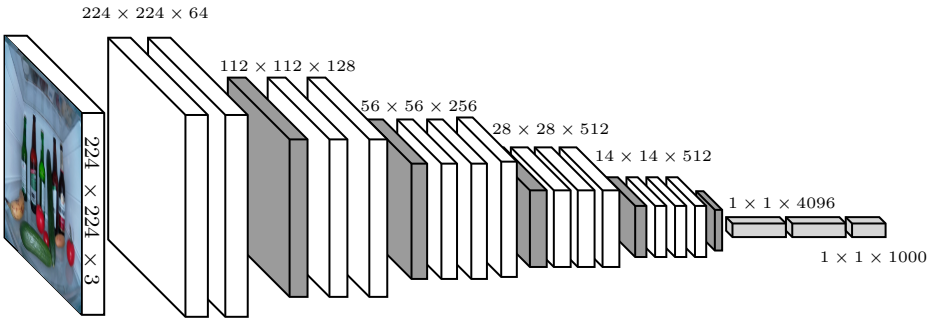


Figure 4.1: Architecture of VGG-16 (without classification layer). White boxes represents convolutional layer, gray boxes represents maximum pooling layers and light gray boxes fully connected layers. Illustration adapted from [49; 16].

several convolutional layers follow, which use a 3×3 kernel. Throughout the network, these convolutional layers decrease in width and increase in depth. Between these convolutional layers are max pooling layers. At the end of this network there are three fully connected layers. In total, the VGG-16 network has 13 convolutional connected layers. Another variant, called VGG-19, uses 16 convolutional layers. This architecture can be used as a feature extractor. To classify images with this network, a classifier layer is necessary. As this architecture uses many convolutional layers, the training of this network is very slow and the weights of the individual layers require a relatively large amount of memory.

4.1.2 Inception V2

Inception V2 was first presented by Szegedy et al. in 2015 [51]. It is the successor to Inception V1, which was presented one year earlier by the same authors [50].

The basic idea of Inception V1 is that an object, from which the features are to be determined, can be found in different sizes or positions on an image. For this reason, choosing the right kernel size used in the convolutional layers is not an easy task. A large kernel is usually used to capture features from a region of larger sizes, while a smaller kernel finds features from smaller regions. Therefore the use of kernels with different sizes is useful to find all important features in one image. However, the simple sequencing of convolutional layers with different kernel sizes leads to a higher computational effort. For this reason, the approach was chosen to run different sized convolutional filters on the same layer in parallel. The authors refer to this approach as an inception module with dimension reduction, which uses convolutional filters of the sizes 1×1, 3×3 and 5×5. These three filters and a 3×3 max pooling layer are executed in parallel. A graphical representation of this architecture is shown in figure 4.2. Since the use of larger filters on a relatively large number of input data leads to a longer run time, 1×1 convolutional filters are placed before the 3×3 and 5×5 convolutional filters to reduce the number of input data. After the 3×3 max pooling layer, there is also a 1×1 convolutional filter.

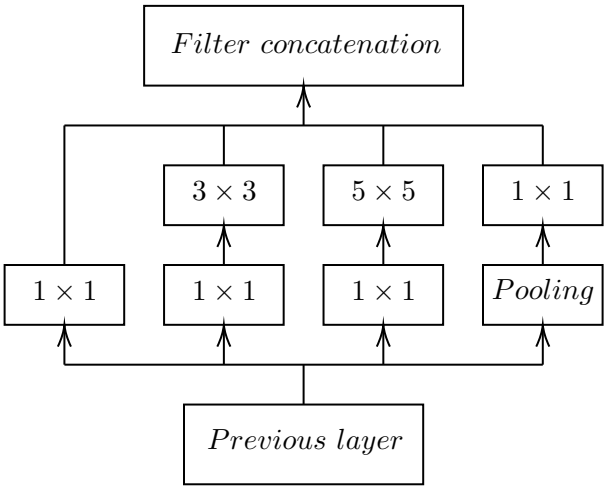


Figure 4.2: Graphical representation of the Inception modul with demension reduction. Adapted from [50].

Inceptions V2 shows some improvements, reduces the number of computations while leading to an enhancement in accuracy. However, the reduction of the dimension leads to a loss of information, which is called *representation bottleneck*. To increase the speed, the 5×5 convolutional filters have been replaced by two 3×3 filters. This is because a 5×5 convolution requires 2.78 times more computational operations than a 3×3 convolution. To further increase efficiency, the $n \times n$ filters were replaced by a combination of $1 \times n$ and $n \times 1$ convolution. For example, the 3×3 filters were substituted by a sequence of a 3×1 and a 1×3 filter, resulting in 33 % less calculations. A graphical representation can be found in Figure 4.3.

4.1.3 MobileNet

The goal of MobileNet V1, which was presented by Howard et al. in 2017, is to minimize the number of computing operations in order to be able to run on mobile platforms [25]. For this reason, the convolutional layer, which requires a relatively large number of computational operations, was replaced by depthwise separable convolutions. A normal convolutional layer gets a $3 \times 3 \times 3$ vector as input. This consists of 3×3 pixels from each of the three color channels. The output has the size of 1×1, which aggregates the values from all input channels. The depthwise separable convolution offers the same functionality, but this one convolution is achieved by several steps, which require less computational effort. The first step is the depthwise convolution. Here, each input channel is considered separately. The input vector of the size 3×3 is reduced to a 1×3 vector for each of the three color channels with the help of a 3×3 kernel. This is followed by the pointwise convolution. This corresponds to a normal convolution with a 1×1 kernel. This way the 1×3 vector is reduced to a 1×1 vector.

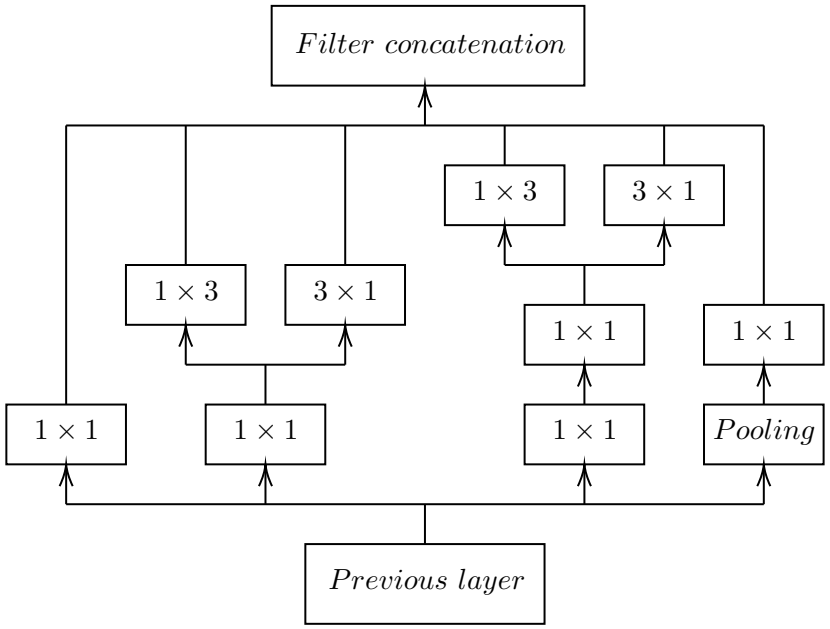


Figure 4.3: Representation of the new architecture of the Inception module used in Inception V2. This example provides the same functionality as the example in figure 4.2, but at lower computational costs. Adapted from [51].

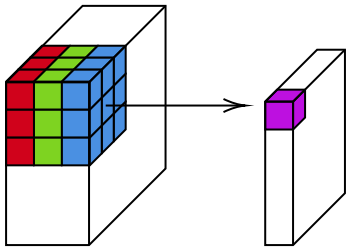
This newly introduced architecture consists of the depthwise and pointwise convolution. After each convolution follows a batch normalization layer and an activation layer. The use of a batch normalization layer reduces the number of training steps by stabilizing the training process [28]. A ReLU6 function is used as activation function. ReLU6 is similar to the normal ReLU function, but the output value for a value greater than or equal to six is also six:

$$ReLU6(x) = \min(\max(0, x), 6).$$

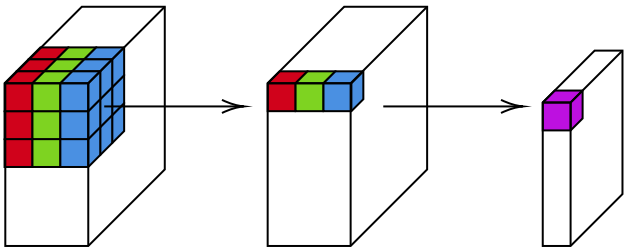
The authors describe, that they use ReLU6 instead of ReLU "because of its robustness when used with low-precision computation" [47]. The entire MobileNet V1 consists of 14 of these blocks, but the input layer is a regular 3×3 convolution layer.

Mobilenet V2

The main goal of Mobilenet V1 was to replace the relatively computationally intensive convolution. However, this also means that the tensors within the network have a comparatively small dimension. This can lead to a loss of information when extracting additional features from these tensors. Therefore slightly different approach was used in MobileNet V2, which was introduced by Sandler et al. in 2018 [47]. In order to combine the speed advantages of small dimensional tensors with the accuracy advantages of higher dimensional tensors, a new architecture has been introduced, called bottleneck residual block. The first component of this block is the expansion layer. This consists of a 1×1 convo-



(a) Convolution using a 3×3 kernel.



(b) Depthwise separable convolution used in Mobilenet V1.

Figure 4.4: Illustration of the depthwise separable convolution used in Mobilenet V1 compared to a regular convolution. Adapted from [24].

lution and serves to multiply the number of output channels. The next layer is a 3×3 depthwise convolution, which was already used in MobileNet V1. To reduce the dimension of the tensor again, the projection layer follows, which is based on a 1×1 convolution. The overall architecture of MobileNet V2 contains 17 of these bottleneck residual blocks, followed by a 1×1 convolution and a pooling layer.

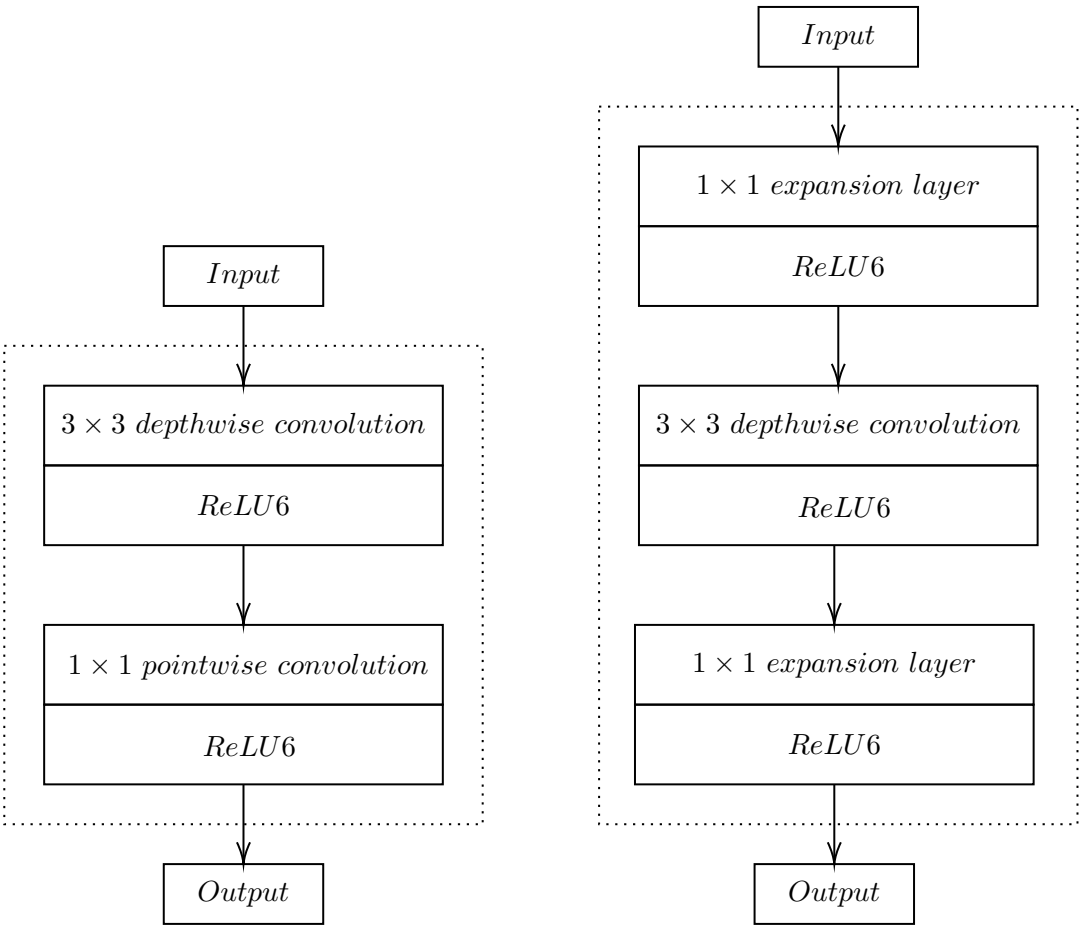
4.2 Region based convolutional neural networks

Faster Region Based Neural Network (Faster R-CNN) is one of the current state of the art methods for object detection and the third object detector based on R-CNN.

R-CNN

R-CNN was first published in 2013 by Girshick et. al. [19]. The original paper shows the significantly higher performance in object detection compared to other established approaches.

R-CNN works by determining different *region proposals*, which are regions of an image that could be part of objects to be recognized. After this first step, each *region proposal* is evaluated by an image classifier. To generate the *region proposals*, *R-CNN* uses the selective search method [56]. This method divides the image in semantic segments by combining parts of an image that have a similar color or texture. For each image, 2000 *category-independent region proposals* are generated. From each of these regions, a



(a) Illustration of the depthwise separable convolution layers used in MobileNet V1.

(b) Architecture of the bottleneck residual blocks used in MobileNet V2

Figure 4.5: Comparisation of the architectures used in MobileNet V1 and MobileNet V2. Adapted from [25] and [47].

4096-dimensional feature map is extracted using five convolutional layers and two fully connected layers. The next step is to check for each of these feature map whether and if so, to which class they belong. During the training of the network, an SVM for each of the classes will be trained, which will be used to evaluate the feature vectors. In the last step, a bounding box is generated for each of the detections if the detection score is above a certain threshold. The disadvantage of this method is the complex training and the slow evaluation speed. All three stages of the network have to be trained independently.

Fast R-CNN

A further development of the R-CNN is the Fast R-CNN, which was published by Girshick in 2015 [18]. The first stage consists of a feature extractor, whose last pooling layer had been exchanged by an *Region of Interest* (ROI) pooling layer. After that, two parallel layers follow, which are used to determine the class of an object and to calculate the bounding box for an object. The feature map is created with the help of a feature extractor. In the implementation provided by Girshick, VGG-16 is used for this. This can also be replaced by any feature extractor. The ROIs are created with a selective search algorithm and the corresponding sections of the feature map are sent to the ROI pooling layer. The ROI pooling layer is used to split the feature map into several smaller feature maps with a fixed size. These sub-feature maps are then sent to the classification and bounding box layers after two fully connected layers. The output of the classification layer is a probability that an ROI can be assigned to a certain class. The bounding box layer is used to further improve the region proposal.

These improvements allow Fast R-CNN to evaluate images 45 times faster than R-CNN. Also the training is 2.7 times faster than R-CNN.

Faster R-CNN

The main difference between Faster R-CNN [41] and its predecessors is the generation of the region proposals. *R-CNN* and *Fast R-CNN* use a selective search algorithm [56] for this task. Due to its long calculation time, it is speed up in *Faster R-CNN* by using a deep fully convolutional network, which is termed as *Region Proposal Network* (RPN). The output of the RPN is a set of regions in which objects could be located. The second module of *Faster R-CNN* is the detector network, which is used to classify the found objects. For this *Fast R-CNN* is used.

Region proposal network

As input, Faster R-CNN receives an image that can have any size. The resolution is then changed to ensure that the largest side is not larger than 1000 pixels and the smallest side is at least 600 pixels in width. Then, a feature map is generated by a feature extractor.

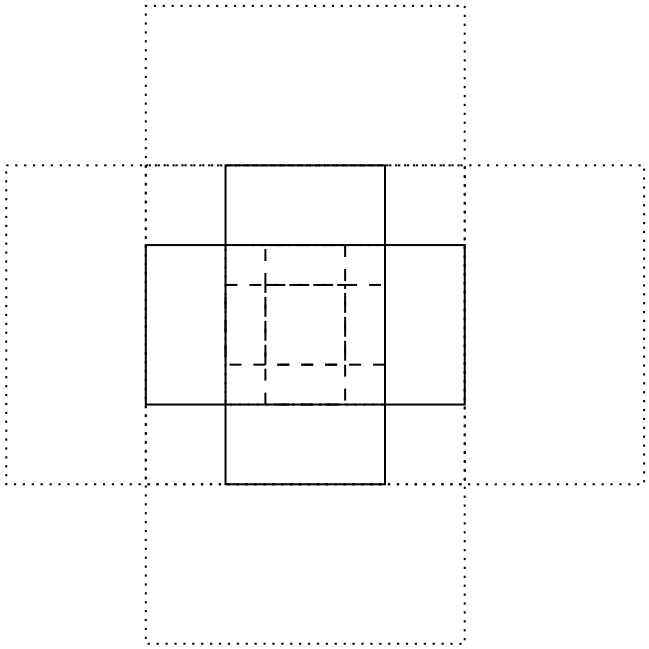


Figure 4.6: Example of the anchor boxes using the aspect ratios 1:1, 1:2 and 2:1 and the scale factors 0.5, 1.0 and 2.0.

Several feature detectors are conceivable here. The authors show that the VGG-16 model could be used for this application. In the next step, the anchor boxes are generated. They differ in size and aspect ratio. In the standard configuration, the side length of a bonding box is 256 pixels. The aspect ratios are 1:2, 1:1 and 2:1. These boxes are then scaled by the size factor (0.5, 1.0, 2.0). An graphical example of this is shown in Figure 4.6. Each box is now arranged on the image at a horizontal and vertical distance of 16 pixels.

Now, the RPN checks which of the anchors enclose an area that contains an object (foreground anchor) or whether it is only background (background anchor). This is done by using a convolutional layer with a 3×3 kernel, followed by two parallel layers with a 1×1 kernel. If a box is a foreground anchor, the edges of the box should be adjusted so that the boundaries of an object are better enclosed by the anchor.

Since many of these anchors overlap, there will also be several proposals for an object that overlap. To ensure that only one proposal remains for an object, non maximum suppression is used. In this process, all proposals for one area are considered and all proposals for which another proposal with a higher score exists in the same place are discarded.

The next step is to assign a class to these region proposals. For this purpose Fast R-CNN is used as a detector. Fast R-CNN uses the already existing feature map. For each region of interest a feature map with a fixed size is extracted from this map. These are then processed in two parallel layer branches. These branches consist of fully connected layers and are used to classify the regions and to adapt the bounding boxes.

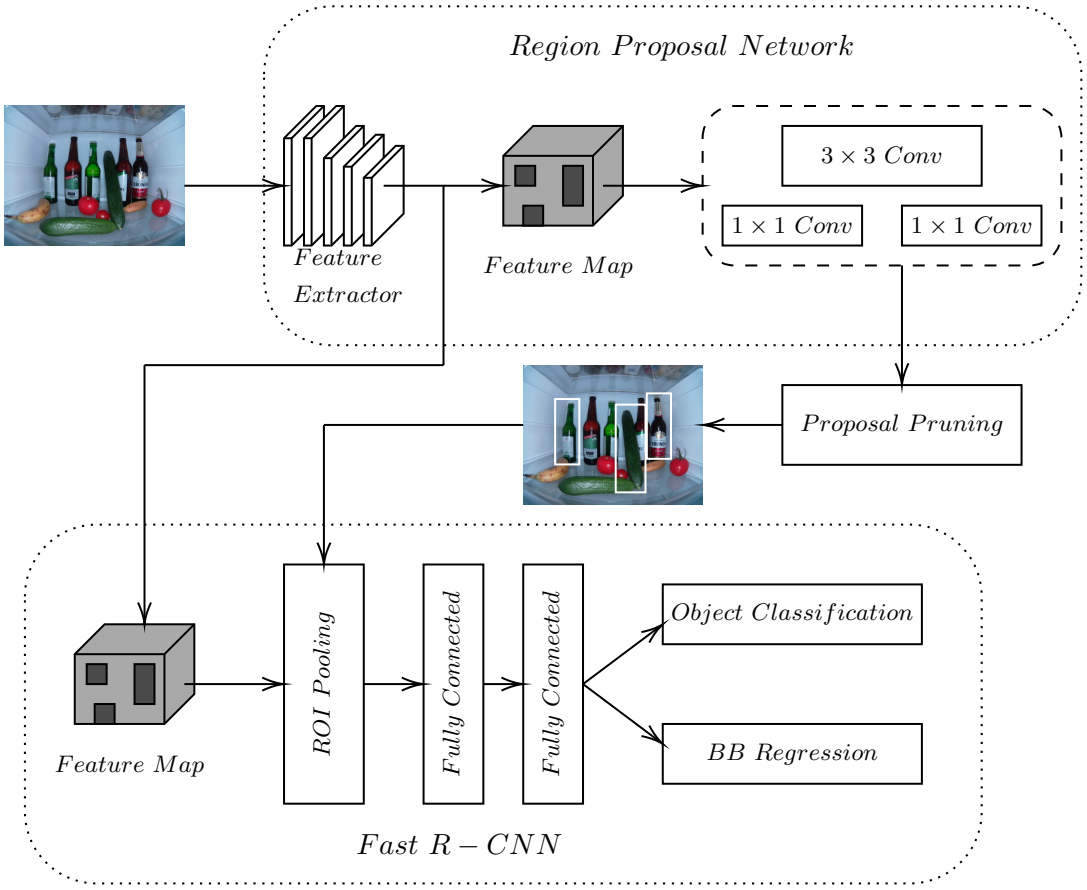


Figure 4.7: Architecture of *Faster R-CNN*.

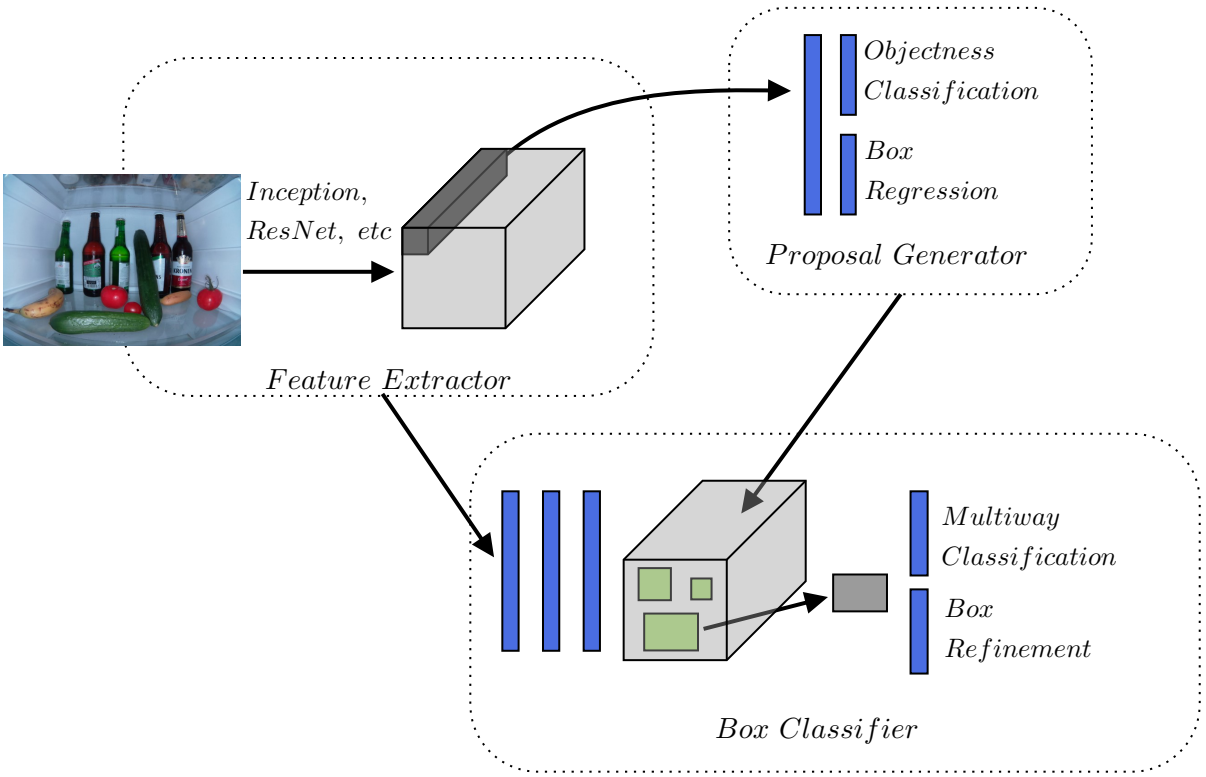


Figure 4.8: Architecture of R-FCN.

4.3 Region based fully convolutional network (R-FCN)

The *Region-based Fully Convolutional Network* (R-FCN) was introduced by Dai et al.[12] in 2016 and uses a similar approach as Faster R-CNN. In the first step region proposals are made by an Region Proposal Network, which uses a fully convolutional architecture. Then, these regions are passed into a classifier network to decide if these regions are showing objects or background. The output of the last convolutional layer is a *position-sensitive score map* for every category. The last layer is a pooling layer, which creates a score for each region. With this score, it is determined if a certain category is present in this region. Figure 4.8 illustrates the architecture of R-FCN.

4.4 Single Shot Multibox Detector (SSD)

The goal in the development of the *Single Shot Multibox Detector* (SSD) was to create an object detector that offers both, high accuracy and high detection speed. In contrast to Faster R-CNN, SSD does not use a region proposal network, thus achieving a higher speed. Object recognition in SSD is achieved by two steps: Feature map extraction followed by convolutional filters to recognize objects. In the first implementation, the authors use VGG-16 as feature extractor, which can also be replaced by any other feature detector. After feature extraction, small convolutional filters are used to determine the

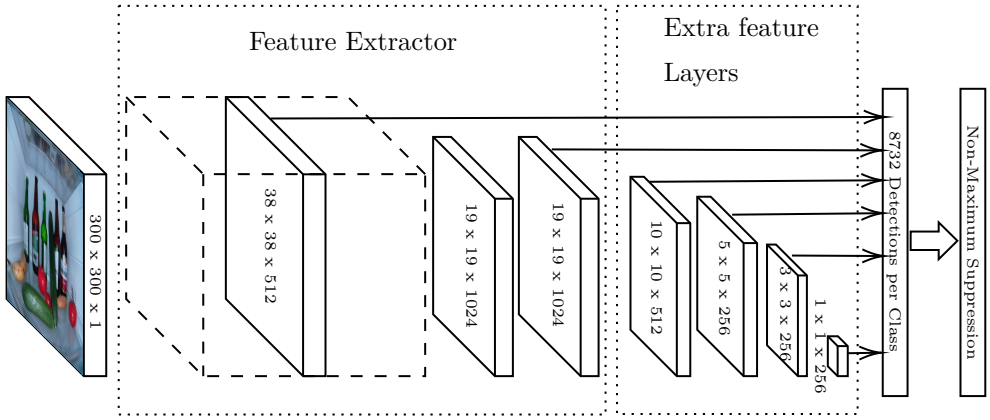


Figure 4.9: Overview of the architecture of SSD [32].

position and class of an object. In the first layer after the feature extractor, the image is split into 38×38 cells. A 3×3 convolution filter is then applied to each cell. The output of this filter is a score for each class as well as the coordinates of a bounding box. Six further layers follow, making a total of 8732 predictions for the whole image. In this process, smaller and smaller feature maps are calculated. Feature maps with a smaller resolution are used to detect larger objects. SSD utilizes non-maximum suppression to filter out different predictions for an object at the same location. If several bounding boxes of the same class overlap, the box with the highest score is selected. In the end, this results in up to 200 predictions for an image.

In addition to SSD, a variant of this object detector was also developed, which is intended for use on mobile devices. This is called SSD Lite [47]. Compared to a normal SSD variant, SSD Lite requires less than one sixth of the required parameters. The authors describe, that this is achieved by replacing all regular convolutions with depthwise separable convolutions with a 1×1 kernel. Nevertheless, the detection performance is only slightly below that of SSD.

4.5 You Only Look Once (YOLO)

An object detector, which was introduced in 2018 by Redmon et al. is called *You Only Look Once (Version 3)* (YOLOv3) [40]. The feature extractor described in the paper is Darknet-53, which uses 53 convolutional layers followed by a normalization layer. As activating function Leaky ReLU is used. In contrast to the previous architectures, YOLOv3 does not use *Pooling layers*. Instead, convolutional layers are used to shrink the feature map. Throughout the network, the feature map becomes increasingly smaller due to the sequence of convolutional layers. In order not to lose any features, three feature maps in different sizes are merged together. This procedure allows for a better recognition of smaller objects.

This is the main difference to SSD. The SSD does not merge several feature maps of different sizes, but tries to classify each feature map.

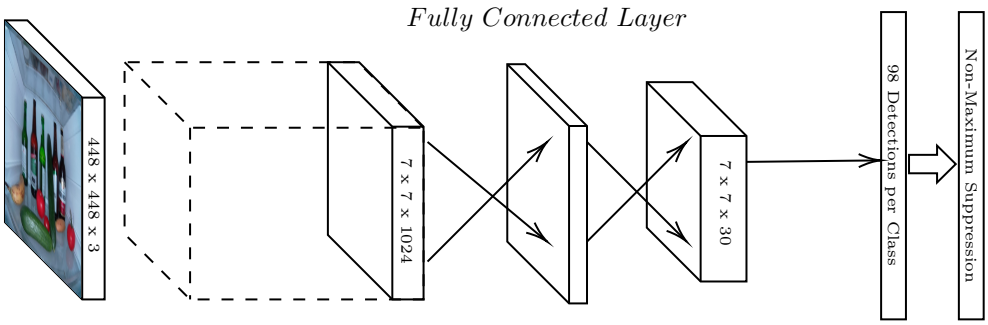


Figure 4.10: YOLOv3 architecture. Adapted from [32].

4.6 Transfer learning

Transfer learning is a widely used technique in the field of neuronal networks. One definition describes transfer learning as "the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned" [55]. During the regular training of a neural network, all weights are randomly initialized. The weights are then changed during the training, so that the output of the network describes the training data as well as possible. For this approach to produce satisfactory results, a large amount of training data is required. Furthermore, the training must be performed with a relatively high number of training steps. If a network has been trained in this way, it can be further trained with a new data set without having to reinitialize the weights beforehand. This allows better results to be achieved, even if the second data set used contains less data. Also, a smaller number of training steps is necessary for continued training. In this context, it is advantageous if the two data sets used for training have certain similarities. In case of image classification, the image classifier could first be pre-trained on a data set with many images of different classes. Then a smaller data set with fewer classes can be used to specialize the network for the second data set.

This procedure has been applied in the process of this thesis. Different object detectors have been chosen, which have already been pre-trained on the MSCOCO data set [31]. This data set contains 2.5 million labeled objects, divided into 91 classes. Subsequently, the object detectors were further trained with the help of the training data set, which is presented in section 5.2.

Chapter 5

Implementation and data set

This chapter describes the used software and frameworks, the used data set and provides details on the implementation and the developed prototype.

5.1 Used frameworks

TensorFlow

TensorFlow is an open source Python library, which was developed by Google in 2015 [4]. It is written in C++ and Python and is mostly used in the field of machine learning. TensorFlow supports a variety of Platforms, such as different CPUs and GPUs, and can even run on mobile devices. It is also available for all major operating systems, like Linux, macOS, Windows and mobile platforms.

Darknet

Darknet is an open source framework, that is written in C and CUDA [39]. Its main purpose is to run neural networks, which are using the YOLO architecture. Darknet supports CPU and GPU calculations.

5.2 Data sets

Within the context of this thesis several image data sets were created. These are used to train the neural networks, to compare the performance of the neural networks and to test and evaluate the developed application.

For the creation of the training data set, it should be taken into account that the food of which the data set consists is typically found in a refrigerator. Therefore, it was initially considered to use images of milk products, such as yogurt or milk cartons. However, it was not possible to find a sufficiently large number of different images that could be used to train a neural network in a reasonable way. There are several image data sets that contain only food. However, these data sets mostly show food that has already been processed and, for example, is served on plates. Since such food is usually not found in a refrigerator, the use of these data sets was also not an option. Therefore a custom data set has to be created. The used pictures are from ImageNet [13]. ImageNet is a large database of images that was developed to provide training images for object recognition projects. The ImageNet database contains more than 14 million images, divided into over 20,000 categories. Another advantage of ImageNet is that the database may be used for non-commercial research and educational purposes.

The next step was to find categories of images that are typically found in the refrigerator and for which enough individual images exist. There is no specific answer to the question how many images are necessary to train an image classifier sufficiently. In this thesis the number was set to at least 1000. This number was chosen because in the *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC), about 1000 images per category will be provided as training data. The goal in the ILSVRC is to develop an object detector with the highest possible accuracy. The first class to be selected are *beer bottles*. This class is interesting for the case of object detection, because these bottles have a unique, standardized shape and differ only in size and color. For the rest of the classes fruit and vegetables have been chosen. It was important that these objects of these classes are available in stores during the whole year, so that the objects needed for the creation of the evaluation data sets can be purchased without any problems. Therefore the following four additional classes were chosen: *bananas*, *broccoli*, *cucumbers* and *tomatoes*. *Bananas* and *cucumbers* were chosen because these two foods look relatively similar and differ only in color and slightly in shape. Among other things, this should be used to test how often *bananas* are confused with *cucumbers* or vice versa. For *broccoli* it is of interest how good the recognition is, because the upper and the lower part of the *broccoli* are optically very different. For *tomatoes*, it was assumed that recognition is relatively easy, as the appearance of individual tomatoes does not differ greatly. Only five classes were chosen, to reduce the training duration. It was decided to take a different number of images for each class, to make possible assumptions on the influence of the number of

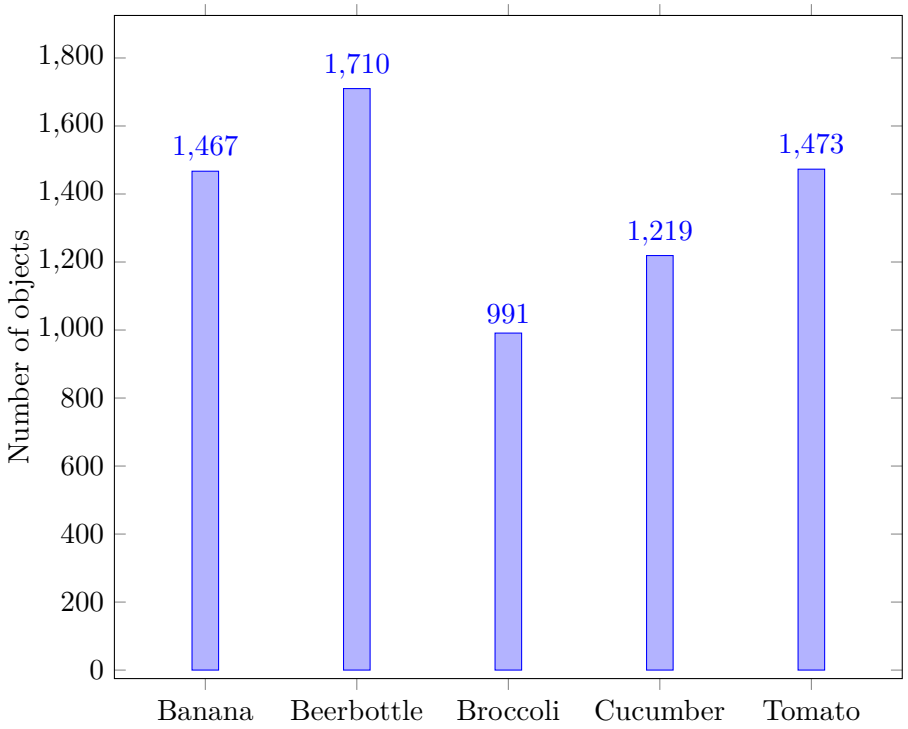


Figure 5.1: Number of objects per class in the training dataset

training images. The whole training data set consists of 6860 objects. Figure 5.1 shows the composition of the data set.

5.2.1 Evaluation data sets

During this master thesis, three data sets were created to evaluate the object detectors. The first data set was used for a proof of concept and contains of 69 images. Therefore different objects of the five classes were randomly arranged on a table and photographed, using a mobile phone camera (Xiaomi Mi 8). Example images of this data set are shown in figure 5.2a. The image resolution is 4032×3024.

For the second data set 166 photos were taken. This time, a wide angle Raspberry Pi camera module is used, which was placed inside a fridge. It was decided to mount the camera on the inside of the door so that the contents of one level of the refrigerator can be captured by the camera. Other positions of the camera can also be considered. For example, the camera could be mounted on the back or side of a compartment. It would also be possible to mount the camera on the ceiling of the refrigerator so that the objects can be seen from above. It was decided against this approach, as the chosen position, which provides a front view, leads to partial covering of the individual foods. Since these masking problems can never be completely avoided, it is important to see what results the tested object detectors provide in such situations. Another important factor is, that the camera

can only be attached at the top in compartments with a certain height. Furthermore, mounting the camera on the inside of the door has the advantage that a certain distance of a few centimeters between the camera and the objects in the refrigerator is always guaranteed. In this way, it is never possible for an object to be positioned so close in front of the camera, that the entire view is obscured.

The resolution for the images in this data set is set to 1920×1080. With these images it has been made sure, that images are created that show only one single object of a class and other images that only show several objects of a single class. Furthermore, there are also images that show objects of all classes in random arrangements. This data set was used to make the first comparisons of the object detectors. It was soon realized that the automatic white balance makes the colors on the images appear very different and that a light source with a neutral white color temperature should be used. Examples of these images can be found in figure 5.2b. The middle image shows a good example of how the white walls of the fridge appear yellowish. It is assumed that this could lead to wrong detections.

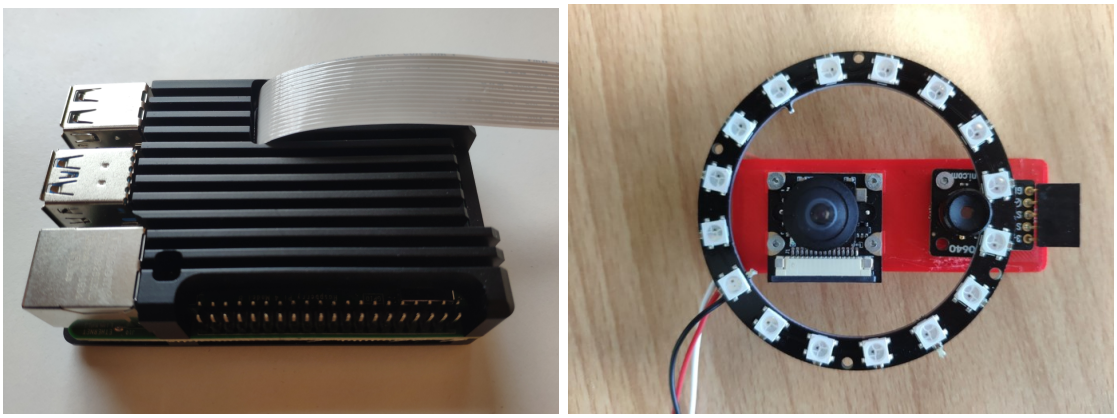
The third data set was also taken using the Raspberry Pi camera module. This time the white balance was set to a fixed value and a different light source was used (see section 5.3). This data set consists of 620 images with a resolution of 2592×1944 pixels. In contrast to the previous data sets, this time different variants of one class were used. Besides normal bananas, mini bananas or green bananas were used. The beer bottles have different sizes and colors. Also the used cucumbers and tomatoes had different sizes. Figure 5.2 shows example images from each of the three data sets.

(a) The images of the **first data set**, which was used for the proof of concept. The objects were placed on a table and photos were taken form different angles.

(b) The images of the **second data set**, were created by placing a Raspberry Pi camera module inside a refrigerator. For these pictures an automatic white balance was used and a LED strip with warm white LEDs was used for illumination. As a result, the walls of the refrigerator appear yellowish in some pictures, while they are white in other pictures.

(c) The pictures of the **third data set** were also taken in a refrigerator with the Raspberry Pi camera module. This time the white balance uses a fixed value and a LED ring was used for illumination, which is described in more detail in section 5.3.

Figure 5.2: Example images from the three evaluation data sets.



(a) Image of the Raspberry Pi 4 B used in this thesis **(b)** Image of the Raspberry Pi camera module

Figure 5.3: Images of the prototype used in the context of this thesis.

5.3 Prototype

The prototype developed in this thesis is based on a Raspberry Pi 4 B and a Raspberry Pi camera module with a wide angle lens. The Raspberry Pi is a single-board computer equipped with a 1.5GHz 64-bit quad core ARM Cortex-A72. For this thesis a version of the Raspberry Pi 4 B with 4GB RAM is used. The operating system used is Raspbian 10, which is based on the Linux kernel 4.19. To reduce latency during read and write processes a 128GB U3 micro SD card is used. The Raspberry Pi camera module offers a maximum resolution of 3280×2464 pixels and is connected to the Raspberry Pi via CSI (camera serial interface). The camera module offers a field of view of 160 degrees. For illumination a ring with 16 WS2812B LEDs is used. The LEDs can be controlled individually and the color can be set for each LED separately. However, in the photos made for this thesis each LED was used with white light. Figure 5.3 shows the LED ring with the camera module in the middle. This arrangement was chosen to achieve the most uniform illumination of the refrigerator and to avoid shadows on the images. It should also be mentioned that the Raspberry Pi is not placed in the refrigerator but is located outside the cooled area and is connected to the camera with a long flat ribbon cable. If the camera remains in the refrigerator for a longer period of time, care should be taken that condensation water does not damage the camera's electronics. To seal the camera, epoxy resin would be a suitable material.

Since all calculations are performed on the Raspberry Pi, the heat development can be quite large. This is especially the case if several images are processed during the evaluation. To prevent the Raspberry Pi from overheating, it is equipped with a passive cooler. In addition to the actual object recognition, the Raspberry Pi should also be able to provide the information obtained to a user. For this purpose, all objects found and the coordinates of the bounding boxes are stored in a database. SQLite is used for this

purpose. A web interface is used to display this data. This is written in Django and allows to view the database in a web browser. To do this, a user device must be connected to the same network as the Raspberry Pi. Storing the detection results in a database has the advantage that further functionalities can be developed independently of the object detection itself. This includes, for example, the integration of a thermal camera or recipe suggestions based on the objects in the refrigerator.

Chapter 6

Metrics used in the evaluation

This chapter describes the metrics that are used in the evaluation of the object detectors. This includes the *Mean Average Precision* (mAP) (see section 6.1), which provides information on the quality of object detection. However, since the correct number of objects to be detected is particularly important in this use case, a new metric has been developed that determines the number of incorrectly classified objects. This type of metric is discussed in section 6.2.

6.1 Mean Average Precision

The mAP [59] is used in many online object detection competitions, such as the *Pascal Voc Challenge* [14; 15], the *Coco Detection Challenge* [1] and the *Open Images Challenge* [2]. For each competition, this metric is slightly modified and adapted to a particular challenge. This master thesis uses the metric as described for the *Pascal Voc Challenge* [3], implemented by Cartucho [9].

The mAP uses the *Intersection over Union* (IoU), also known as the Jaccard index, to calculate the overlap between two bounding boxes. To determine whether a detection is correct or false, the ground truth bounding box B_{gt} and the predicted bounding box B_p are required. Now the *Intersection Over Union* is calculated by

$$IoU = \frac{B_{gt} \cap B_p}{B_{gt} \cup B_p}.$$

An illustration of the IoU ca be found in Figure 6.1.

A detection is considered as *True Positive* (TP) when the classification is correct and the IoU is greater or equal than a predefined threshold. This threshold is set to 50 % when the *Pascal Voc Challenge* metric is used. A detection is wrong and counted as a *False Positive* (FP), when the IoU is smaller than the threshold. A result is considered *False Negative* (FN) if an object is not recognized.

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union} =$$

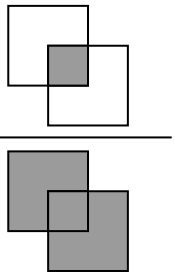
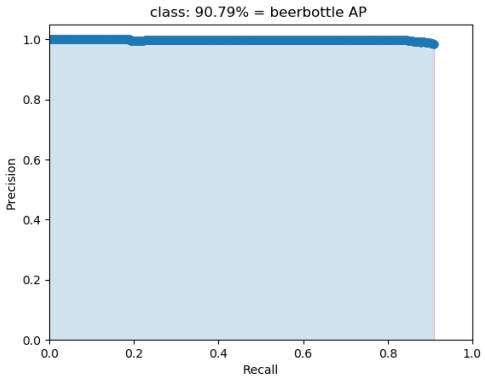
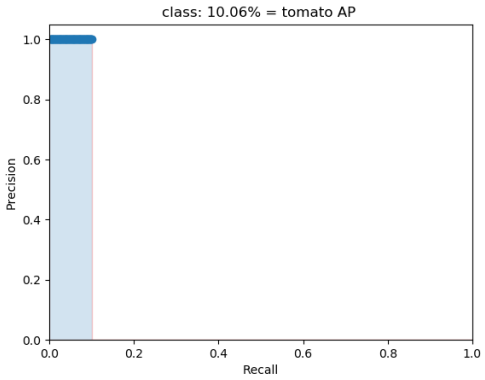


Figure 6.1: Graphical representation of the calculation of the IoU.



(a) Precision × Recall of the class beerbottle.



(b) Precision × Recall of the class tomato.

Figure 6.2: Examples of Precision × Recall curves. These curves are generated using the results of Faster R-CNN Inception V2 on the third data set.

The ability to recognize only relevant objects is called precision. This is calculated by

$$Precision = \frac{TP}{FP + TP}.$$

Recall denotes the property of finding all ground truth bounding boxes, which is determined by

$$Recall = \frac{TP}{TP + FN}.$$

This means, that the *Recall* is higher, the lower the number of False Negatives is. The lower the number of False Positives, the higher the *Precision*. Now the *Precision* × *Recall* curves are created by plotting the precision and recall values for every class. An example is shown in Figure 6.2. In the next step, the area under the Precision × Recall curve is calculated. This value is considered as *Average Precision* (AP). Finally the mAP is calculated from the arithmetic mean of the AP-values of all classes.

6.2 Custom metric

For the application case described in this master thesis, namely the detection of food inside a refrigerator, the correctly detected number of objects is more important than the exact

position of the detected objects. Therefore, another metric is introduced to compare the number of objects detected with the number of objects actually in the refrigerator. For this purpose, the number of objects for each class is determined for every image from the evaluation data set. Then, for each image, the number of objects of a class recognized and the percentage value is determined. This value is now summed for all images for each class and the arithmetic mean is calculated.

Chapter 7

Evaluation of the detection results

There is a variety of object detectors that use neural networks. In this chapter the different object detectors will be compared. Afterwards, it will be examined which of the detectors is best suited for the intended application. The aim is to find a detector that offers an optimal trade-off between accuracy and speed. In addition, the selected detector should be able to run efficiently on an embedded system, such as a Raspberry Pi.

7.1 Literature review

In recent years, object detectors which use neural networks have become increasingly powerful. In the context of this thesis, some of these detectors have been examined to see if they are suitable for this application. In summary, the following detector architectures were examined:

- Faster R-CNN [41]
- R-FCN [12]
- SSD [32]
- YOLOv3 [40]

The direct comparison on the basis of the corresponding publications is difficult, since in these papers the main focus lies on the used metric like the mAP. The problem here is that the data sets used in the different papers for training and evaluation are different [57]. It is also rarely discussed how much memory is used and how good the performance is on mobile devices. It is also difficult to compare, as various parameters differ from paper to paper, such as the framework used or the size of the input image [26].

The *TensorFlow Model Zoo* [53] offers a variety of pre-trained object detection models implemented in TensorFlow. For each provided model it is indicated how long the

Table 7.1: Comparison of different models, which are available in the TensorFlow Model Zoo and were compared during this thesis. These models have been trained and evaluated using the COCO data set [53]. The inference time was calculated using images with a resolution of 600 x 600 pixels and the inference was done with a Nvidia Titan X.

Architecture	**Feature extractor**	**Inference Time (ms)**	**mAP**
Faster RCNN	NAS	1833	43
Faster RCNN	Inception Resnet v2	620	37
SSD	Resnet 50	76	35
Faster RCNN	Resnet 101	106	32
Faster RCNN	Resnet 50	89	30
RFCN	Resnet 101	92	30
Faster RCNN	Inception v2	58	28
SSD	Inception v2	42	24
SSD	Mobilenet v2	31	22
SSD Lite	Mobilenet v2	27	22
SSD	Mobilenet v1	30	21

inference of an image with a resolution of 600 x 600 pixels using an Nvidia GeForce GTX TITAN X takes. In addition, the mAP of each model is displayed when using the COCO data set. In the following, only models that output a bounding box for recognized objects were considered. Table 7.1 provides an overview over different models.

These values suggest that *Faster R-CNN NAS* and *Inception Resnet V2* give the most accurate results, but are also the slowest. If they are run on an embedded system, this speed disadvantage could be significantly increased. A compromise between speed and accuracy is shown by detectors that use *Resnet* or *Inception* as feature extractor. Detectors using *Mobilenet* show the lowest accuracy but also the highest speed.

The paper by Liu et al. [32], shows a different result. Here the three architectures Faster R-CNN, YOLO and SSD were compared, utilizing the VOC2007 data set. VGG-16 was used as the feature extractor for all three networks. SSD performs best with a mAP of 77.2 %. Faster R-CNN and YOLO only achieved a mAP of 73.2 and 63.4 respectively. SSD shows the best results in both categories speed and accuracy. It should be noted that the three networks compared receive images with a different resolution as input. Further results of this experiment are shown in table 7.2.

Table 7.2: Comparison of Faster R-CNN, SSD and YOLO according to [32]. The FPS were calculated using a Nvidia Titan X for inference.

Architecture	VOC2007 test mAP	FPS	Input resolution
Faster R-CNN	73.2	7	1000 x 600
YOLO	63.4	45	448 x 448
SSD300	79.8	46	300 x 300

7.2 Training on evaluation data set

The next step is to train these pre-trained object detectors with the training data set presented in subsection 5.2.1.
Faster R-CNN NAS is not used in this thesis, because the training requires a graphics card with more than 11 GB of RAM. Furthermore the inference time of *Faster R-CNN NAS* is almost 2 seconds using a TITAN X GPU, so the processing of an image on an embedded system would take much more time.
A PC with an Intel Core i7-6700k @ 4 GHz, a Nvidia GeForce RTX 2080 Ti (11 GB) and 32 GB RAM is used for the training. The training parameters used are the settings as described in sample configuration files [54]. The aspect ratios of the anchors are set to 0.5, 1.0 and 2.0 and the scales to 0.25, 0.5, 1.0 and 2.0. The number of steps is limited to 200,000, while the batch size is set to 1 for all *Faster RCNN*-networks. The batch size for *SSD*-based detectors is set to 2. Setting this value to 1 respectively 2 for all detectors is not possible, otherwise the training is terminated by a failure of TensorFlow. YOLOv3 was trained with 7000 iterations.

7.3 Results using the first data set

As proof of concept and to get an overview of how the chosen object detectors perform, the first evaluation series is performed on the first evaluation data set. The performance is only compared with the use of the mAP metric. The results can be found in Table 7.3. The confidence threshold for the evaluation is set to 50 %. These results show that the performance of the Faster R-CNN detectors are very similar. The mAP of these ranges from 52.72 % to 65.09 %. The same is true for R-FCN Resnet101, whose mAP reaches a value of 58.43 %. The results of the SSD-based detectors are also very close in comparison. The mAP for these models ranges from 19.02 % to 37.13 %. The SSD Lite MobileNet reaches the highest value. YOLOv3 shows similar results as the other one-stage detectors, with an mAP of 35.36 %. The distribution of the individual AP values for the separate classes is also very similar for the Faster R-CNN detectors. *Bananas* achieve the highest AP of 81.26 % to 84.71 % with these detectors. The SSD based detectors in this category

only achieve an AP of 6.21 % to 39.53 %. The detectors based on MobileNet V1 show a very poor detection rate for *bananas* with an AP of 6.21 % respectively 8.33 %. Considering the AP value for *beer bottles*, it is noticeable that these are best detected by Inception V2 with distance, which achieves a mAP of 79.02 %. The other detectors show much lower results in this category. Especially remarkable is the value for SSD MobileNet V1, which indicates that not a single bottle is detected. SSD MobileNet V1 FPN shows a slightly better result in detecting *beer bottles* with an AP of 1.92 %. The results for the class *broccoli* are much higher in the case of SSD based object detectors. In some cases these values are even higher than those of the other detectors.

Cucumbers show an AP of about 80.43 % and 80.04 % with Faster R-CNN and R-FCN, respectively. For the SSD based detectors this value ranges from 56.52 % to 60.87 %, except for MobileNet V1 FPN and MobileNet V2, where this value is 15.68 % and 5.33 % respectively.

Summing up, it can be concluded that the one-stage detectors reach the lowest performance, irregardless of its variants. Why this is the case can not be explained with absolute certainty. A possible assumption would be that a larger amount of training data is beneficial for this detector. However, this does not apply to SSD Lite, as it shows the highest mAP value of the SSD based detectors. Similarly good results can also be achieved with SSD Inception or SSD Resnet50, respectively. The YOLOv3 object detector shows similar results as the SSD-based detectors. However, the values show a more even distribution.

The detectors that use two-stage detection show the best overall results. In terms of individual categories, Faster R-CNN Resnet101 achieves the highest AP value in three classes. Considering the mAP, Faster R-CNN Inception V2 achieves the highest value of 65.09 %.

Table 7.3: Results of the evaluation using the first data set.

Architecture	**Feature Extractor**	**Average Precision**					**mAP**
		Banana	Beerbottle	Broccoli	Cucumber	Tomato	
Faster R-CNN	Inception V2	83.27	**79.02**	58.40	76.09	28.68	**65.09**
	Resnet101	**84.71**	58.96	**67.27**	**80.43**	21.71	62.62
	Inception Resnet V2	81.26	50.89	38.02	**80.43**	12.81	52.72
SSD	Inception V2	26.49	25.40	48.17	59.45	16.28	35.16
	Mobilenet V1	8.33	0.0	30.77	58.14	13.61	22.17
	Mobilenet V1 FPN	6.21	1.92	28.00	15.68	**43.27**	19.02
	Mobilenet V2	14.45	52.66	15.88	5.33	21.90	22.05
	Mobilenet V2 (SSDLite)	34.86	17.46	54.31	56.52	22.48	37.13
	Resnet50	39.53	5.67	52.30	60.87	18.60	35.39
R-FCN	Resnet101	79.71	61.27	50.53	80.04	20.59	58.43
YOLOv3	Darknet53	28.59	53.35	27.00	31.56	36.31	35.36

7.4 Evaluation on the second data set

Based on these results, the proof of concept is considered successful. Since Faster R-CNN Inception gives the best results, the first step is to make changes to this detector. The other detectors should also be evaluated on this data set to see if other detectors on another data set show different results. In the next step, new images are created for a second data set. This data set contains 166 images and is intended to evaluate first modified network architectures. For this purpose, new images are taken inside the refrigerator. For illumination, a LED strip with warm-white LEDs is used, which was placed above the objects to be photographed. It quickly became clear, while creating the data set, the white balance and exposure were automatically set for each new image. Nevertheless, the photos were not discarded, as it is still interesting to see how the object recognition behaves when the images do not use a uniform white balance and exposure time. In the further development process, it is decided to limit the selection of object detectors and to continue to examine only those detectors that have shown the best results so far. To simplify further tests and evaluations, it is decided to focus on a single framework. Therefore only detectors based on TensorFlow are evaluated in the subsequent sections. Those are three versions of Faster R-CNN: Inception V2, Inception Resnet V2 and Resnet101, as well as SSD Inception V2 and R-FCN Resnet101.

For the *banana* class the best values are found in Faster R-CNN Inception V2 and Inception Resnet V2. The AP for these classes are 49.61 % and 61.24 % respectively. The other detectors achieve much lower results for this class. For SSD Inception V2 this value is only 5.10 %. The detection of the *beer bottles* shows significantly better AP values for all detectors. With Faster R-CNN Inception V2 the AP reaches a value of 92.37 %. The lowest result in this category is achieved by SSD Inception V2. In the category *broccoli* the best detection is found with Faster R-CNN Inception Resnet V2. This is at 87.51 %. The worst result shows R-FCN Resnet101 with an AP of 53.78 %. For the *cucumbers* the Faster R-CNN-based object detectors show the best performance. The AP value is 81.74 % with Inception Resnet V2 the highest. For Inception V2 and Resnet101 the AP score is 67.27 %. SSD Inception V2 only reaches a value of 12.73 % in this category. In the *tomato* category the best results are attained by Faster R-CNN Inception V2. The AP here is 49.30 %. The lowest AP is found with R-FCN Resnet101 with 3.52 %. The highest mAP is found with Faster R-CNN Inception Resnet V2 and is 70.81 %. The second highest mAP is 66.00 % of Faster R-CNN Inception V2. The lowest mAP of the tested object detectors, with a value of 31.69 %, is reached by SSD Inception V2.

Table 7.4: Results of the second data set regarding the AP and mAP.

Architectur	**Faster R-CNN**			**SSD**	**R-FCN**
Feature Extractor	**Inception V2**	**Inception Resnet V2**	**Resnet 101**	**Inception V2**	**Resnet 101**
Banana	49.61	**61.24**	19.72	5.10	101
Beerbottle	**92.37**	80.63	89.15	51.14	82.94
Broccoli	71.47	**87.51**	71.64	66.26	53.78
Cucumber	**67.27**	81.74	65.45	12.73	52.73
Tomato	**49.30**	42.96	21.13	23.24	3.52
mAP	66.00	**70.81**	53.42	31.69	41.00

7.5 Evaluation on the third data set

The third and final data set is created to simulate a real environment and to evaluate the performance of the object detectors using a larger data set in order to obtain more significant results. The pictures therefore contain several objects of one class in different variations. This time, care is taken that the white balance is set to a fixed value. More detailed information can be found in subsection 5.2.1. Afterwards one of these detectors is selected and several methods are tested to see whether the detection performance can be increased even further.

In order to evaluate the performance of the object detectors considered so far, only the mAP or AP of the individual classes is examined first. According to this metric, the following object detectors are selected for a comparison and a detailed evaluation based on the third data set:

- R-FCN Resnet101
- Faster R-CNN Resnet101
- Faster R-CNN Inception Resnet101
- Faster R-CNN Inception V2

These detectors have now been evaluated using the third data set. The results are shown in Table 7.5. All tested detectors show a similar performance with a mAP of 31.74 % to 41.46 %. The highest mAP is achieved by Faster R-CNN Inception V2. Considering the AP for *bananas*, the values range from 30.43 % to 34.83 %. In this case, the highest value is reached by Faster R-CNN Inception Resnet V2. The AP for *beer bottles* shows that Faster R-CNN Inception V2 reaches a value of over 90 %. The other detectors only reach values between approximately 60 % and 70 %. *Broccoli* is also best identified by Faster

Table 7.5: Evaluation on the third data set using the mAP-metric.

Architecture		**R-FCN**	**Faster R-CNN**		
Feature Extractor		Resnet 101	Resnet 101	Inception Resnet V2	Inception V2
Average Precision	Banana	30,43	38,97	**42,23**	34,83
	Beerbottle	59,65	67,21	62,90	**90,79**
	Broccoli	16,22	32,78	30,82	**39,29**
	Cucumber	45,20	39,46	**50,17**	32,32
	Tomato	7,23	8,00	2,20	10,06
mAP		31,74	37,28	37,67	**41,46**

R-CNN Inception V2. The AP has a reading of 39.29 %. For *cucumbers*, the highest AP is 50.17 % achieved by Faster R-CNN Inception Resnet101. The other detectors are 5 to 15 percentage points below this value. The detection of *tomatoes* is, as in the previous tests, one of the most difficult disciplines. Here, the highest AP of Faster R-CNN Inception V2 is achieved with 10.06 %.

Based on these results, Faster R-CNN Inception V2 is selected because it has the highest mAP and the highest AP in two classes. Furthermore, it is taken into account that the later object detector should be able run on an embedded platform. There is limited computing capacity on an embedded system, so an object detector should be selected to ensure the fastest possible evaluation speed. Looking at the values in the Table 7.1, Faster R-CNN shows the fastest evaluation speed. This theory is also supported by the results in [26]. Faster R-CNN also shows the fastest evaluation time of the object detectors considered here. This is due to the fact that Inception V2 only requires 10.1 million parameters. Resnet101 requires 42.6 million parameters and Inception Resnet V2 even 54.3 million parameters.

Now that Faster R-CNN Inception V2 has been selected as the object detector, the next step is to try to increase performance even further. The first simple approach here is to extend the training time. The training is extended by 50 %, to a total of 300,000 steps. It quickly becomes apparent, that this attempt has not led to a significant improvement. The mAP only increases by 0.52 %, but the AP decreases in two categories. As other papers suggest [46], a too extensive training of neural networks, can lead to overfitting, which causes a decrease in accuracy.

Freezing layers

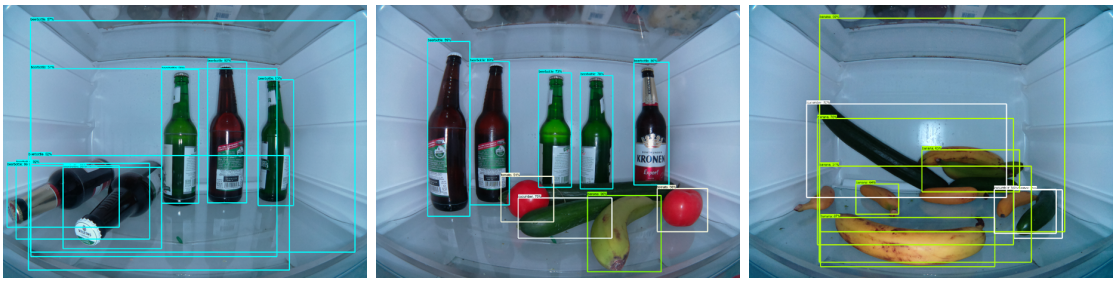
In the next experiment, an approach is chosen that tries to use the fact that the object detectors are already pre-trained on the COCO data set. Here, the weights of certain layers are frozen before the network is trained. An approach that is used in [30], and has produced good results, freezes the weights of all convolutional layers. The weights of the classification layer are trained again, using the a different data set. For the experiments in this thesis, the weights of all *FeatureExtractor* layers are frozen, so only the *ClassPredictor* and *BoxEncodingPredictor* layers are trained. The results can be found in Table 7.6.

Considering the results regarding AP and mAP values, they show a clear increase: The AP for *bananas* increases by 16.70 % and the AP for *beer bottles* by 7.05 %. The AP for *broccoli* is enhanced by 15.17 %. The biggest enhancements are found regarding AP for *cucumbers*, which is increased by 28.38 % and the AP is heightened by 19.57 %. The mAP is increased by 17.37 %.

At first these results seem to be very good, but when looking at the bounding boxes of the detected objects and the values of the custom metric, it becomes clear that many objects have been detected that are not present on the images. Looking at the values of the custom metric, 6.87 % of the detected *bananas* are non-existent. The unmodified version of Faster R-CNN does not misdetect an object in this category. During the evaluation of the entire data set, Faster R-CNN Inception V2 detects 13 *beer bottles*, one *broccoli* and two *cucumbers* that do not exist. In the version of Faster R-CNN where the layers are frozen, the numbers are significantly higher. Among the detected objects, there are also 10.66 % *beer bottles*, 6.71 % *broccoli* and 3.47 % *cucumbers* that are not found on the images. In absolute numbers, this corresponds to 73 *bananas*, 128 *beer bottles*, 19 *broccoli* and 19 *cucumbers*. These results are shown in Table 7.7. Both versions of the detector did not detect any *tomatoes*, which are not in the image. Examples of the bounding boxes generated by both version of Faster R-CNN Inception V2 can be found in Figure 7.1. It is important to note that this behavior only occurs with some images. In some images the objects are well detected, even better in comparison to non modified version of Faster R-CNN, in other images several objects are detected that are not present in the images. The bounding boxes of the non-existent objects have different sizes and can often be found where the background area of the image actually is. In most cases, false detection only occur, if an object of the wrong identified class is on the image. Other factors that are responsible for the detection of non-existent objects cannot be clearly defined.

After this attempt to freeze only the weights of the classification layer, the *Mixed_5c* layers, which corresponds to the last network layers before the classification layers, are frozen additionally. The results of this attempt, however, show significantly worse values than in the version that is not modified. This resulted in a mAP of only 21.86 %. Detailed

(a) Faster R-CNN Inception V2 (Non modified version)



(b) Faster R-CNN Inception V2 (Frozen feature extractor layer)

Figure 7.1: Sample images that visualize the Faster R-CNN Inception V2 detections. In the top row are the detections generated by using the unmodified version of Faster R-CNN Inception V2. The images in the bottom row show the detections of Faster R-CNN Inception V2 where the feature extraction layer is frozen before training.

Table 7.6: Results of the first modified version of Faster R-CNN Inception V2 using the third data set.

Model	**Faster R-CNN Inception V2**			
Modification	Non modified	300k steps	Frozen feat. ex.	Frozen 5C layer
Banana AP	34,83	37,58	**51,53**	13,84
Beerbottle AP	90,79	91,40	**97,84**	45,64
Broccoli AP	39,29	37,10	**54,46**	18,37
Cucumber AP	32,32	36,55	**60,70**	17,66
Tomato AP	10,06	7,27	**29,63**	13,81
mAP	41,46	41,98	**58,83**	21,86

Table 7.7: Number of non-existent objects that are detected in percent.

Model	**Faster R-CNN Inception V2**			
Modification	Non modified	300k steps	Frozen feat. ex.	Frozen 5C layer
Banana	0.00	0.00	**6.87**	0.00
Beerbottle	1.08	2.50	**10.66**	0.42
Broccoli	0.35	0.00	**6.71**	0.00
Cucumber	0.37	0.18	**3.47**	0.37
Tomato	0.00	0.00	**0.00**	0.00

Table 7.8: Overview of the different scales and aspect ratios.

Model	**Scales**	**Aspect ratios**
Anchors1	0.25, 0.5, 0.75, 1.0, 2.0	0.5, 1.0, 2.0
Anchors2	0.25, 0.5, 1.0, 2.0	0.25, 0.5, 1.0, 2.0
Anchors3	0.25, 0.5, 1.0, 2.0	0.125, 0.25, 0.5, 1.0, 2.0, 3.0, 4.0
Anchors4	0.25, 0.5, 1.0, 2.0	0.0625, 0.125, 0.5, 1.0, 2.0, 4.0, 5.0
Anchors5	0.25, 0.5, 1.0, 2.0	0.5, 1.0, 2.0, 3.0
Anchors6	0.25, 0.5, 1.0, 2.0	0.5, 1.0, 2.0, 4.0

values can be found in Table 7.6 and Table 7.7. Based on these results, the approach of freezing certain layers is not followed.

Modifying anchor boxes

The next approach that is tested is to optimize the generation of the anchor boxes. These anchor boxes are used by the RPN of Faster R-CNN (see section 4.2). In the first step, these boxes are distributed over the entire image. Then, it is checked whether the area enclosed by the box shows an object or parts of the background. If an object is detected, it is then classified. In the default configuration, which was presented in the paper for Faster R-CNN, the boxes have aspect ratios of 1:2, 1:1 and 2:1. For the scales, sizes of 0.5, 1.0 and 2.0 are specified. In the example configuration files for TensorFlow, an additional anchor box with scale 0.25 is given and was kept for the following experiments. An attempt has now been made to adapt these aspect ratios to the objects to be detected. An overview of this can be found in table 7.8.

When adjusting the boxes to the objects to be recognized, it is tried to adapt them to the specific objects. Therefore, it is advantageous to have a square box with the aspect ratio 1:1 to recognize *tomatoes* and *broccoli*. That is why this aspect ratio is used in all attempts. For the other three classes elongated boxes are more suitable. Depending on

Table 7.9: Results of Faster R-CNN Inception V2 with modified anchor aspect ratios regarding the AP and mAP.

	Average precision in %					
Faster R-CNN Inception V2	**Banana**	**Beerbottle**	**Broccoli**	**Cucumber**	**Tomato**	**mAP**
Non modified	34,83	**90,79**	39,29	32,32	10,06	41,46
Anchors1	1,37	34,73	39,60	0,73	0,00	15,29
Anchors2	43,54	76,65	28,84	43,10	7,20	39,87
Anchors3	**48,41**	85,36	31,39	38,26	7,60	42,20
Anchors4	39,70	75,85	33,04	39,07	11,84	39,90
Anchors5	45,00	77,83	**48,05**	**52,36**	**25,60**	**49,77**
Anchors6	41,99	79,32	42,37	45,70	10,32	43,94

Table 7.10: Results of Faster R-CNN Inception V2 with modified anchor aspect ratios regarding the number of detections of non-existend objects in percent.

	Number of non-existend objects in %				
Faster R-CNN Inception V2	**Banana**	**Beerbottle**	**Broccoli**	**Cucumber**	**Tomato**
Non modified	0,00	1,08	0,35	**0,37**	0,00
Anchors1	0,00	0,08	0,00	0,00	0,00
Anchors2	0,19	0,67	0,00	0,00	0,00
Anchors3	**0,38**	0,75	0,35	0,18	0,00
Anchors4	0,09	**2,08**	0,00	0,00	0,00
Anchors5	0,19	0,25	**0,71**	0,18	0,00
Anchors6	0,19	2,00	0,35	0,00	0,00

the orientation of the *bananas*, *beer bottles* or *cucumbers*, boxes with an aspect ratio of 1:2, 1:3, 1:4 or 1:5 might give better results. It should be noted that an increase in the number of boxes can also lead to a longer execution time. In the following tests, the configuration of the anchor boxes is adjusted and different versions of Faster R-CNN Inception V2 were trained. The results can be found in table 7.9 and table 7.10.

For the Anchors1 model, it will be tested whether the accuracy of the detection can be further increased by introducing an additional anchor box with a different scale. Here the scale 0.75 was chosen, which is between the previous used sizes. The aspect ratios of

0.5, 1.0 and 2.0 are unchanged. The results show, that the mAP is significantly reduced and is now only 15.29 %. By introducing this aspect ratio, no more *tomatoes* could be detected. The detection of *bananas* and *cucumbers* is also significantly worsened. The AP for these classes is now only 1.37 % and 0.73 % respectively. Also the AP for the *beer bottle* class is with 34.73 % clearly below the AP of the unmodified version. Only the AP for *broccoli* is increased by 0.31 percentage points. Based on these results, it is decided not to change the aspect ratio in the following attempts and to keep the standard sizes of 0.25, 0.5, 1.0 and 2.0. The Anchors2 model should be used to test whether the detection of elongated objects in a horizontal position can be increased. Therefore, the aspect ratio of 0.25 is added to this model. The results show, that the mAP has only deteriorated by 1.59 percentage points. The AP for the *banana* and *cucumber* classes are increased by 8.71 and 10.78 percentage points respectively. It is therefore reasonable to assume, that the introduction of this elongated anchor box will make it easier to detect objects such as *cucumbers* or *bananas*, as long as they are in a certain position.

For the Anchors3 model, the number of anchors boxes with an elongated aspect ratio are increased in order to further improve the detection rate in the *banana*, *beer bottle* and *cucumber* categories. Therefore, the additional aspect ratios 0.125, 3.0 and 4.0 are introduced for the next experiments. The results show, that the AP in the categories *bananas* and *beer bottles* is increased by 4.87 and 8.71 percentage points respectively, compared to Anchors2. However, the introduction of the new anchor boxes also leads to a deterioration of 3.84 % in the *cucumber* category.

A similar approach to Anchors3 is followed for the anchor configuration of Anchors4. To further adapt the anchors to *beer bottles* or *cucumbers*, anchors with aspect ratios 5.0 and 0.0625 are used. The aspect ratios 0.25 and 3.0 were not used in this trial. The results of this experiment indicate that this adaptation to specific objects did not work. The achieved AP values and mAP are almost identical to those of Anchors2. Interestingly, the AP for *tomatoes* is increased to 11.84 %, which is even higher than the AP of the unmodified version. The Anchors5 version gives the best results for the mAP, which is 49.77 %. In this version, the aspect ratio of 3.0 was chosen as addition to the three standard aspect ratios, since, as explained above, most objects have an elongated shape. This improvement is particularly noticeable in the *cucumber* category, which has an AP of 52.36 %. Also the AP value for *broccoli*, with a value of 48.05 % could be increased significantly. This could be explained by the fact that a *broccoli* photographed from the side can be well captured by an anchor box with an aspect ratio of 3:1. The same is true for *bananas*. Compared to the unmodified version, the AP is increased by 10.17 %. What is particularly noticeable here, is the AP value for *tomatoes*, which is 25.60 %. This is significantly higher than the values of the other models. However, this significant increase cannot be explained by the newly introduced aspect ratio.

Table 7.11: Comparisation of the AP and mAP of Faster R-CNN Inception V2, using differend confidence thresholds.

		Average precision in %					
Faster R-CNN Inception V2	**Threshold**	**Banana**	**Beerbottle**	**Broccoli**	**Cucumber**	**Tomato**	**mAP**
Non modified	50%	34.83	90.79	39.29	32.32	10.06	41.46
Anchors5	50%	45.00	77.83	48.05	52.36	25.60	49.77
Anchors5	30%	48.05	83.55	50.36	57.84	29.17	53.79
Anchors5	20%	49.56	86.43	53.66	60.82	30.38	56.26
Anchors5	10%	51.68	89.32	54.94	64.29	34.09	58.26
Anchors5	5%	53.84	90.93	56.20	66.70	36.17	60.77

Another attempt to improve the aspect ratio in order to improve detection is made with the Anchors6 model. Due to the relatively good results shown by the Anchors5 model, this time an aspect ratio of 1:4 is used instead of the previously used ratio of 1:3, as it is assumed that this ratio could better represent the objects used. The results still show a significant improvement over the unmodified version with a mAP of 43.94 %. However, this version shows no improvement over Anchors5. Only the AP of *beer bottles* is 1.49 % higher than the AP of Anchors5.

In summary, the adaptation of the anchor boxes to a particular class seems to work partially. Although, increases in the AP values of individual classes cannot be properly explained by the introduction of new aspect ratios. It should be noted here, that the anchor boxes used do not have to enclose the object to be detected with absolute precision. Since the anchor boxes are only used to check whether an object is located in a certain area or not, parts of an object can also be located outside this box. If it is then decided that the anchor box encloses an object, it is adapted even more precisely to the object.

Reduction of the confidence threshold

Based on these results, the Anchors5 version is chosen as it has the highest mAP and the highest AP in three categories. In comparison to the unmodified version, all the values are significantly higher, except for the AP for *beer bottles*. Several tests with a reduced confidence threshold are now being performed. In the previous tests, an object is considered to be detected if the confidence was above 50 %. In the subsequent tests, this threshold is reduced and it is tested how this change affects the results. The can be seen in table 7.11 and in table 7.12.

Table 7.12: Comparisation of the number of non-existend objects that were detected, using differend confidence thresholds.

		Number of non-existend objects in %				
Faster R-CNN Inception V2	**Threshold**	**Banana**	**Beerbottle**	**Broccoli**	**Cucumber**	**Tomato**
Non modified	50%	0.00	1.08	0.35	0.37	0.00
Anchors5	50%	0.19	0.25	0.71	0.18	0.00
Anchors5	30%	0.28	0.58	1.77	0.18	0.13
Anchors5	20%	0.47	1.08	1.77	0.37	0.13
Anchors5	10%	1.04	1.58	3.53	1.10	0.77
Anchors5	5%	2.17	2.41	4.95	1.65	1.55

The results show that the reduction in the confidence threshold of the AP increases significantly in all categories. This results in an increase in the mAP of about 4 to 11 percentage points. These results for the mAP are thus about 12 to 29 percentage points higher than those of the unmodified version of Faster R-CNN Inception V2. In the *beer bottle* category, the AP also significantly increases and now shows similar values to the AP of the unmodified version. Looking at the values of the custom metric, however, it becomes apparent that there are also more detections of non-existent objects (see table 7.12). To analyze these values, the bounding boxes on the images of the evaluation data set are examined. These values are highest for *broccoli*. This can be explained by the fact that, due to its appearance, a *broccoli* can also be seen as a composition of several smaller *broccoli*, which are also detected at a lower confidence threshold. Looking at the detection results of *beer bottles*, it is often the case that two *beer bottles* standing close together are detected as three bottles. This also applies to *bananas*. A good compromise of a relatively high mAP and few false detections is shown by the version Faster R-CNN Inception V2 where anchor boxes with the aspect ratios 1:1, 1:2, 1:3 were used and the confidence threshold is set to 20 %.

So far, only the mAP and the numbers of detections, that are not incorrect have been considered in the evaluation. Table 7.13 shows the percentages of correctly detected objects for each class. There, all results for models considered so far, can be found. Due to the formula used to calculate this metric, these values are very similar to the AP values considered previously. If these values of the unmodified version are compared to those of Anchors5_20, it can be seen, that the detection rate in four of the five classes has been increased, in some cases significantly. For the classes *bananas* and *broccoli*, the number of correctly classified objects was increased by 15.82 % and 14.13 % respectively. In contrast, the number of correctly classified *beer bottles* decreased by 5,16 %. In the case

of *tomatoes*, this value was increased by 21,42 % and reaches a values of 31,48 %. This value is significantly higher than that of the other models tested. The highest increase can be observed for the class *cucumbers*. The detection rate is now 63.25 %, which is an increase of 30.53 %.

Table 7.13: Number of correct detected objects in %.

Model	Modification	Banana	Beerbottle	Broccoli	Cucumber	Tomato
RFCN Resnet101	-	33.62	58.53	16.96	46.25	7.23
Resnet 101	-	40.30	67.78	33.22	40.22	8.00
Inception Resnet V2	-	42.66	63.20	32.16	51.92	2.45
Inception V2	-	36.53	90.84	40.99	32.72	10.06
Inception V2	300k	38.89	92.01	37.10	36.75	7.74
Inception V2	Frozen	54.71	91.34	50.53	58.50	29.81
Inception V2	5C	14.60	45.88	18.73	17.55	13.94
Inception V2	Anchors	1.69	34.72	40.99	0.73	0.00
Inception V2	Anchors2	45.20	76.69	30.39	43.51	7.61
Inception V2	Anchors3	50.19	85.18	31.10	39.31	7.74
Inception V2	Anchors4	41.43	76.02	33.92	39.67	12.00
Inception V2	Anchors5	47.18	77.44	49.82	53.56	26.19
Inception V2	Anchors6	43.79	79.68	43.11	45.70	10.32
Inception V2	Anchors5_30	50.47	82.76	51.24	60.15	29.81
Inception V2	Anchors5_20	52.35	85.68	55.12	63.25	31.48
Inception V2	Anchors5_10	56.40	88.68	54.77	66.18	35.23
Inception V2	Anchors5_05	59.23	89.93	55.48	68.37	37.42

7.6 Background subtraction

The results so far indicate that the accuracy of the selected object detectors can be significantly increased by the applied modification, which are discussed in the previous sections. The best results for the tested data set can be achieved by adding an anchor box with the additional aspect ratio of 1:3 and setting the confidence threshold to 20 %.

Even if the number of correctly detected objects has increased significantly, there are still objects that can not be detected by the detector. Therefore this approach alone is not sufficient to be used for object detection in a refrigerator.

In the remaining process, a new approach will be chosen, which is adapted to the specific refrigerator scenario. A refrigerator is typically filled with food, for example after a grocery purchase, which is then gradually removed. In some cases, the removed objects are placed back in the refrigerator. Also, it will often be the case that a new purchase is made and new food items are placed in the refrigerator before it is completely empty. So how exactly the amount of food in a refrigerator changes varies from person to person. To simplify this scenario, it is first assumed that only a few parts of the contents change. This means that either a small number of objects are removed or added again.

To find out what has changed in the refrigerator, background subtraction is used. In this procedure two images are taken and one image is subtracted from the other. This makes it easy to see which part of the image has been changed. The approach is to take a picture of the current contents of the refrigerator as soon as the door of the refrigerator is opened and then closed again. An illustrative example of this process is shown in figure 7.2. Figure 7.2a shows the state before removal. In figure 7.2b, the banana has already been removed. Figure 7.2c shows, the result of the background subtraction. Here, it is very clear to see in which area the image has changed. The coordinates of this area can now be determined very easily and a classification can be performed on this area.

In this thesis a pipeline is developed, which combines the previously optimized object detector with the procedure of background subtraction. An illustration of this can found in 7.3. For this purpose it is assumed that the current contents of the refrigerator are unknown. An image of this state is called I_{t_0}. The following image, where something has changed, has the designation I_{t_1} and presents the current state of the refrigerator.

On picture I_{t_1} in the first step an object recognition is performed by Faster R-CNN. All detected objects and their bounding boxes are saved. Then a background subtraction is performed with the images I_{t_0} and I_{t_1}. The coordinates of the areas in which something has changed are also temporarily stored. Now, all areas where the Faster R-CNN object detector has found an object are colored black. In the next step, all areas that are found with the help of the background subtraction are cut out of the image I_{t_1} and another object detection by Faster R-CNN is performed on these areas. The black coloring of the

(a) Image I_{t_0}, which represents the initial status.

(b) Image I_{t_1}, where a mini banana was added.

(c) Visualization of the background subtraction.

Figure 7.2: Example of the background subtraction. It is easy to see, where something was chanced in the image.

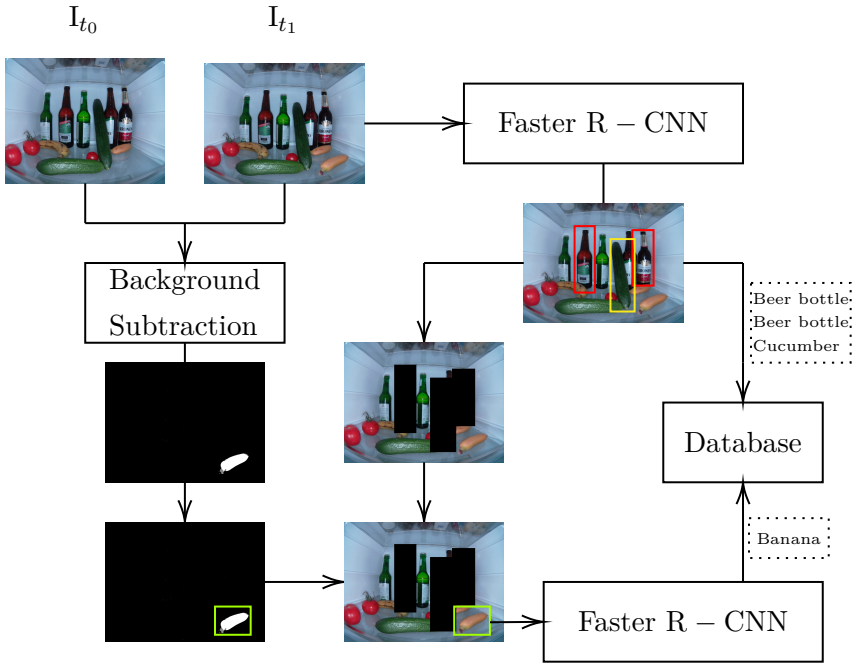


Figure 7.3: Illustration of the developed pipeline, which combines the background subtraction with an object detector.

objects already found is intended to prevent objects from being detected more than once by Faster R-CNN during the subsequent re-run.

Initially, it was considered to use a regular image classification, such as an Inception image classifier, on these image sections instead of the second run of Faster R-CNN, as this could save time. However, this idea was rejected due to the fact, that if several objects are placed back in the fridge and they are too close to each other, several objects may be located in one section. A regular image classifier would not give useful results for an image with several objects.

One difficulty that arises when using background subtraction is setting the threshold. This value specifies how far the pixels of I_{t_1} must deviate from I_{t_0} at a certain position in order to this new region to be recognized by the background subtraction. If this value is set too high, no area is recognized by the background subtraction. If this value is too low, the section in which a new object is located can be divided into several parts. This can either lead to no object is detected, or to multiple detections of a single object. This threshold value can also be different for individual objects. For example, a *beer bottle* may be cut out as a single object, but a *broccoli* will be divided into many small sections. In order to get a greater benefit from the information of the background subtraction, it is decided to use the position of the areas that are obtained by the background subtraction of the images I_{t_0} and I_{t_1}, not only for the evaluation of I_{t_1}, but to use these areas also in the evaluation of the following images I_{t_n}. This has the advantage that if, for example, an object in I_{t_1} was only recognized by background subtraction, it can also be recognized in I_{t_2}. To optimize the speed, the first entries should be deleted after a certain number of runs.

To evaluate this procedure, a synthetic data set is created. For this purpose, objects are cut out of different images and inserted into an image showing an empty refrigerator. This creates a sequence of images in which a new object is added to each image. An exact evaluation of this procedure is difficult, because there are many scenarios that can occur. These are for example adding one or more objects, removing one or more objects, changing the position of certain objects or changing the complete content, as it can happen after a purchase. For simplicity, only one or two objects are added when creating the image sequences. This is also because background subtraction is not expected to give better results when removing objects. In the following, the presented procedure for one of these sequences is explained in detail. The results for each image of the sequence can be found in 7.14.

The first picture shows a *beer bottle*, which is recognized by the first stage. Since this is the first image of the sequence, no background subtraction can be performed yet. On the second image a *broccoli* was added. This can not be detected by the first detector stage. The *beer bottle* was recognized by the first detector stage. By the second stage the *broccoli* was detected. So, all objects on the second image could be identified correctly. In the third image a *banana* was inserted. This is not recognized by the first stage of the object detector. The second stage recognizes this *banana* as two bananas. Interestingly, the *broccoli*, which was not detected by the first stage in the second image, is correctly detected in this run. Also in the following images this *broccoli* can already be detected by the first stage detector. The fourth image is extended by a *tomato*. This can only be classified by the second stage. The phenomenon that occurred in the third image for the *broccoli* now also appears for the *banana*. This *banana* can be detected by the first stage detector on the fourth and on the following images. Added to the fifth image is a

Table 7.14: Detailed results from one run of the final object detector. For this evaluation a synthetic data set was used. For each image in the sequence it is listed which objects were found by the first run of Faster R-CNN (1st stage) and the second run in combination with the background subtraction (2nd stage). The objects that are in the image are called groundtruth (GT).

Image		**Number of objects per class**				
		Banana	Beerbottle	Broccoli	Cucumber	Tomato
	1st stage	0	1	0	0	0
	2nd stage	-	-	-	-	-
	GT	0	1	0	0	0
	1st stage	0	1	0	0	0
	2nd stage	0	1	1	0	0
	GT	0	1	1	0	0
	1st stage	0	1	1	0	0
	2nd stage	2	1	1	0	0
	GT	1	1	1	0	0
	1st stage	1	1	1	0	0
	2nd stage	1	1	1	0	1
	GT	1	1	1	0	1
	1st stage	1	1	1	0	1
	2nd stage	1	1	1	0	1
	GT	1	1	1	1	1
	1st stage	1	1	1	0	1
	2nd stage	1	2	1	0	1
	GT	1	2	1	1	1

cucumber. It cannot be detected on this image or on the following images either by the first stage detector or by the second stage detector. On the sixth image a further *beer bottle* was inserted. This can only be classified by the second stage detector. In summary, it can be concluded for this example that the combination of Faster R-CNN and the background subtraction resulted in an increase of the detected objects.

7.7 Evaluation of the detection speed

When measuring the run time, it is assumed that the resolution of the images has a high influence on it. Therefore, it is first tested whether the use of images with a lower resolution leads to worse results. However, it has to be considered that if an image is used as input for a Faster R-CNN, it is first scaled to a new size. The width will not exceed 1024 pixels and the height 600 pixels. For this trail, Faster R-CNN Inception V2 Anchors5 with a threshold of 20 % is used, as this showed the best results so far. The results can be found in 7.15. The original images have a resolution of 2592×1944 pixels. For the tests, the images are scaled to the following resolutions: 2000×1500, 1500×1125, 1000×750 and 500×375. It can be seen that a reduced input size leads to a worse recognition rate, for most classes. However, these values only differ from the original values by a few percentage points. Interestingly, the reduction of the image size even leads to a slightly higher recognition rate for the *cucumber* class. The greatest deviation is achieved when an input resolution of 500×375 is used. Also the number of incorrectly classified objects is only very small.

In the next step the run time of the developed pipeline is measured using Faster R-CNN in combination with the background subtraction, which was presented in the previous section. This pipeline is evaluated on a Raspberry Pi 4. The times were measured at different points of the pipeline and are shown in Figure 7.4. At the beginning of the pipeline a TensorFlow session is created. This creation takes about 6 seconds. Afterwards, the image I_{t_1} is copied into a NumPy array. This step is only necessary because this evaluation uses previously saved images and does not directly capture them with the camera. This is where the largest run time differences due to the image size become apparent. For an image with a resolution of 2592×1944 this step takes 13.70 seconds on average. For a resolution of 500×375 it takes only 0.49 seconds.

Table 7.15: Number of correct and false classified objects using Faster R-CNN Inception V2 Anchors5 (Threshold: 20%) with different input resolutions.

Input resolution	**Number of correct identified objects in %**				
	Banana	Beerbottle	Broccoli	Cucumber	Tomato
500px	24.76	79.77	48.76	54.48	17.29
1000px	45.48	84.93	53.36	65.81	23.10
1500px	49.34	83.85	54.06	66.54	25.94
2000px	50.85	85.60	54.06	66.00	28.00
2592px	52.35	85.68	55.12	63.25	31.48
	Number of false identified objects in %				
500px	0.00	0.83	0.35	2.74	0.26
1000px	0.09	0.50	0.71	1.83	0.00
1500px	0.09	0.58	1.06	1.10	0.00
2000px	0.28	0.83	1.41	0.91	0.00
2592px	0.47	1.08	1.77	0.37	0.13

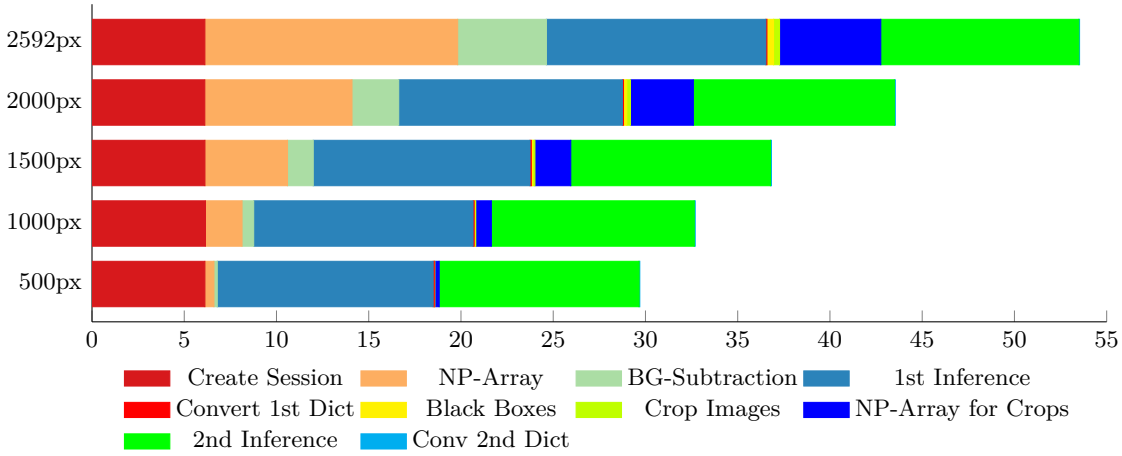


Figure 7.4: Speed measurements of the pipeline using different input sizes.

In the next step the background subtraction is performed. Again, the influence of the image size is very noticeable. At the highest resolution, the duration is 4.81 seconds; at the lowest resolution it is only 0.18 seconds. Subsequently, the first image is evaluated by Faster R-CNN. This takes 11.87 seconds on average and is not dependent on the size of the input image. To process the output of the inference, 0.006 seconds are needed (Convert 1st dict). In the following steps, the black boxes are placed over the image and the image sections are cut out of I_{t_1}. These sections are then converted into a NumPy array again. At this point, the influence of the image size can be seen again. At the highest resolution

Table 7.16: Run time of the developed pipeline using different input resolutions.

Input resolution	Run time in seconds
500px	29.63
1000px	32.63
1500px	36.75
2000px	43.48
2592px	53.47

this step takes 5.48 seconds, at the lowest resolution only 0.23 seconds. After that, these image sections are evaluated by the second run of the R-CNN Faster. This step takes an average of 10.85 seconds for all tested resolutions. In the last step only the output of the second inference is processed. The total run time for an image with the resolution 2594×1944 is 53.47 seconds. The remaining run times can be found in table 7.16.

As already mentioned, the first conversion of the image into a NumPy array is only necessary, if the image is not directly captured by the camera but has to be loaded from the SD card. However, it is also possible to capture an image and receive it directly as a NumPy array. By omitting this conversion, about 13 seconds can be saved at a resolution of 2594×1944. This would result in a run time of about 40 seconds.

Chapter 8

Discussion

In the course of this thesis an object detector based on Faster R-CNN was developed. The results of the evaluation show that this architecture is suitable to detect and classify a variety of objects. Due to the specific improvements, the performance has increased significantly. It should be considered that the training duration and the number of images are limited. However, this does not automatically mean that the recognition performance can be improved by using more images and longer training. As already described in [46], longer training duration can lead to overfitting, which may even result in poorer detection results. Nevertheless, there are still objects that are not recognized. To solve this problem, Faster R-CNN was combined with background subtraction to create the detector described in 7.6. This makes it possible to detect other objects that would not have been detected by the regular version of Faster R-CNN. A phenomenon observed in the evaluation of the background subtraction is, that some objects may not be detected by Faster R-CNN. But if other parts of the image change, some of these objects will be detected, even if the position of the objects has not changed. No explanation for this behavior has been found yet. One difficulty that will always be present when objects are detected, regardless of the context, is the masking problem. This means that objects cannot be detected at all or only partially, which makes classification difficult. Also if a small object is placed in front of a larger object, it is possible that the smaller object is not recognized as an object, because the larger object, which is in the background, influences the separation. No general statements can be made for this masking problem. For example, it depends on the type and arrangement of the objects, how many percent of an object must be visible for it to be identified as an object. The background is also a factor that can influence the identification.

If the detector is to be expanded for the detection of new object classes, new technical difficulties will occur. Apart from the amount of new training data required, the detection of objects that look very similar is difficult. This is especially true for packaged food. The exact contents can only be identified by the writing on the object. An example of this is

yogurts in different flavors or different types of packaged cheese. A similar problem occurs when objects are placed in the refrigerator that are packaged in a freshness container. In this case no statement can be made about their contents. Obviously it is not possible to detect objects for which the detector has not been trained.

In the following, it will be discussed which overall statements can be made about the recognition of the selected classes. First of all, it should be considered whether the number of images used for the training has an influence on the later recognition. For this purpose, the numbers from table 5.1 are considered, as well as the results from chapter 7. The objects where the most difficulties in recognition have been encountered are *tomatoes*. The number of training images for this class is 1,473, almost the same as for *bananas*. The recognition of the *bananas* showed much better results in the evaluation. For the classes *broccoli* and *cucumber* there is less training data than for *tomatoes*. The difference between the number of classes *broccoli* and *tomato* is 482. Therefore there are about 48 % more pictures in the *tomato* than in the class *broccoli*. However, the results of the evaluation show that objects of the classes *broccoli* and *cucumber* are better recognized by most of the tested object detectors than *tomatoes*. Consequently, it is not assumed that the amount of training data is responsible for the poor detection rate of *tomatoes*. The results also show that the *beer bottles* are best detected and sometimes a very high AP of over 90 % is achieved. Since this class also had the most training data, a correlation between the number of training data and the good detection rate cannot be completely excluded. Since there is only about 14 % less training data for the classes *bananas* and *tomato*, this fact is not considered the main reason. It is rather assumed that there is another explanation for this. As already described in section 3.2.1, *tomatoes* seem to have few features that allow them to be identified. This reason could also cause *tomatoes* to be poorly recognized by neural networks.

An event that has not been considered so far is that food is taken out of the refrigerator and put back after a short time. This is the case with jam, cheese or milk, for example. To display the current status of the refrigerator, this situation does not play a special role. If, however, it is necessary to additionally store when an object has been placed in the refrigerator, problems will arise at this point, as it is not possible to decide whether an object was only outside the refrigerator for a short time or whether it has been used up or replaced by a new identical food. However, knowing when a food item was placed in the refrigerator could be of great advantage to the user, as it is less likely that food will spoil before it is consumed.

For the same reason, it would also be an advantage if the specific expiration date of each food could be tracked. For this application, food can be divided into two different classes. On the one hand, foods that have a specific expiry date printed on them and, on the other hand, foods that do not have a specific expiry date. Examples of the first class are yogurt, milk or packaged cheese. The second class includes fruit or vegetables. In

order to automatically exclude the expiry date of first class objects in the system, a camera based character recognition system could be used. However, the technical implementation could prove to be very difficult. Firstly, there is no uniform position where the expiry date is attached. An expiration date can be located at the bottom of a product, even on the bottom side. Even if the date is always in the same place, for example on the top of the packaging, automatic recognition is still difficult. Another reason why it is difficult to detect is that the expiry date printed on the product is sometimes difficult to detect, even for humans. For the second class, however, a statement about the expiry date could be made. To do this, a database would first have to be created, which would provide an approximate shelf life for most fruits and vegetables. With this information, a user could then be informed if a food product was about to expire.

The most important point that should be investigated in this thesis is whether object recognition can be executed on an embedded system. It was shown that the developed pipeline can reach a run time of 40 seconds. Therefore, it is concluded that this goal has been achieved and that the use of an embedded system, such as a Raspberry Pi, is possible and can be used for a consumer product.

This has the advantage that the evaluation is possible locally and no cloud service is necessary to perform the evaluation and provide the results to a user. This guarantees the privacy of the data, as the data is only stored in the Raspberry Pi. A user can also access this data while on the move if the Raspberry Pi is connected to the Internet and has a fixed IP or DNS name.

Chapter 9

Conclusion & Outlook

In this thesis a prototype based on an embedded system should be developed, which is able to recognize the contents of a refrigerator. For this purpose, different data sets for training and evaluation were created. Afterwards different object detectors were trained with the help of these data sets and subsequently evaluated. Initially, the mAP metric was used for this purpose, as it is widely used for comparing object detectors. Since the number of detected objects is much more important than the exact localization for the use case considered in this thesis, a new metric was developed which counts the detected objects and compares this number with the number of objects on the image. Using this metric, it has been noticed that object detectors with a high mAP sometimes detect many objects that are not present on the image. Based on the results of the first test, Faster R-CNN Inception V2 was selected as the best of the object detectors and different approaches were tested to further increase the detection rate. For this purpose anchor boxes with new aspect ratios were used and the confidence threshold was reduced. To increase the detection rate even further, a pipeline was developed that uses background subtraction. This allows the detection rate for individual objects placed in the refrigerator to be increased even further. The speed of the developed system was then tested on a Raspberry Pi. The run time of the pipeline is about 40 seconds. Since no real-time detection is necessary in this application case, it is concluded that an embedded system can be used in this scenario.

One point that makes the detection of objects more difficult is the masking problem. This makes the detection of objects problematic, if parts of an object are obscured by other objects. To further investigate this problem, the use of multiple cameras could be investigated in the following work. The detection rate could also be increased by several images of an object, even if an object is not obscured. The use of text recognition or a camera-based bar-code detector could also be tested to better detect packaged food.

Appendices

Feature detection

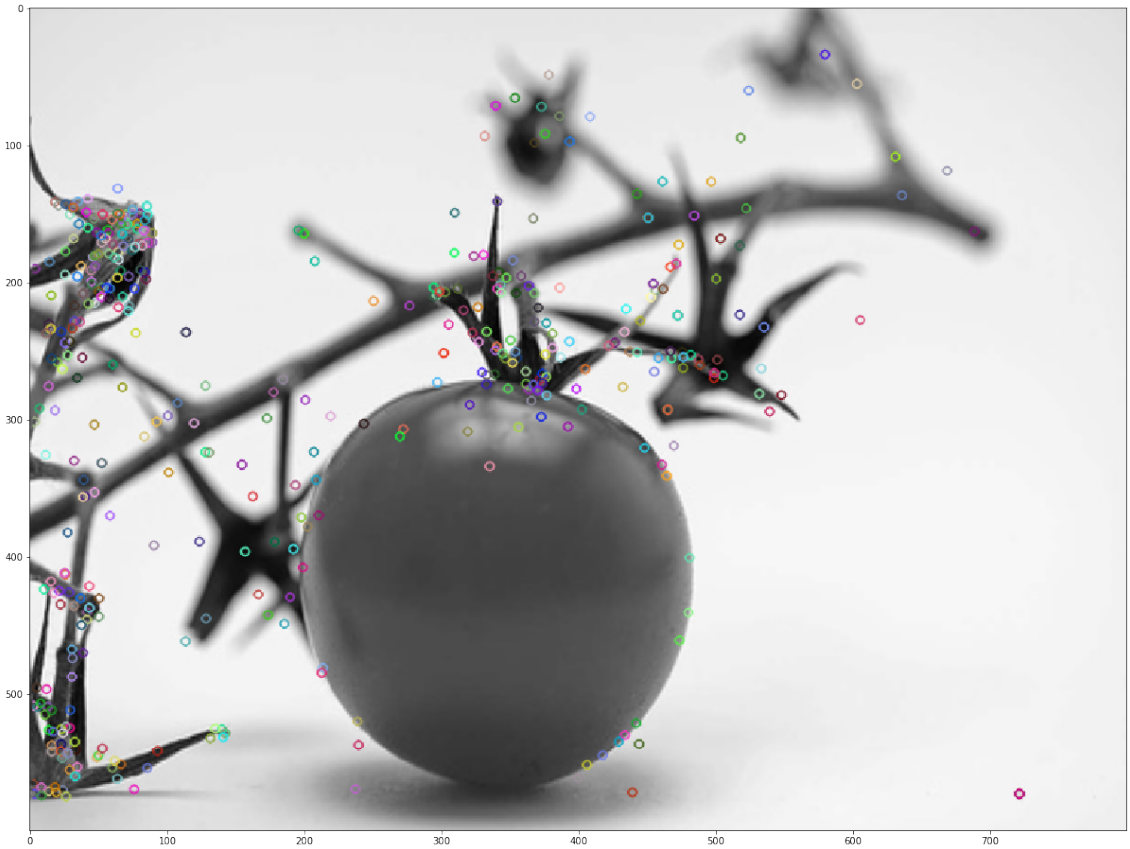


Figure 1: Detected features using SIFT.

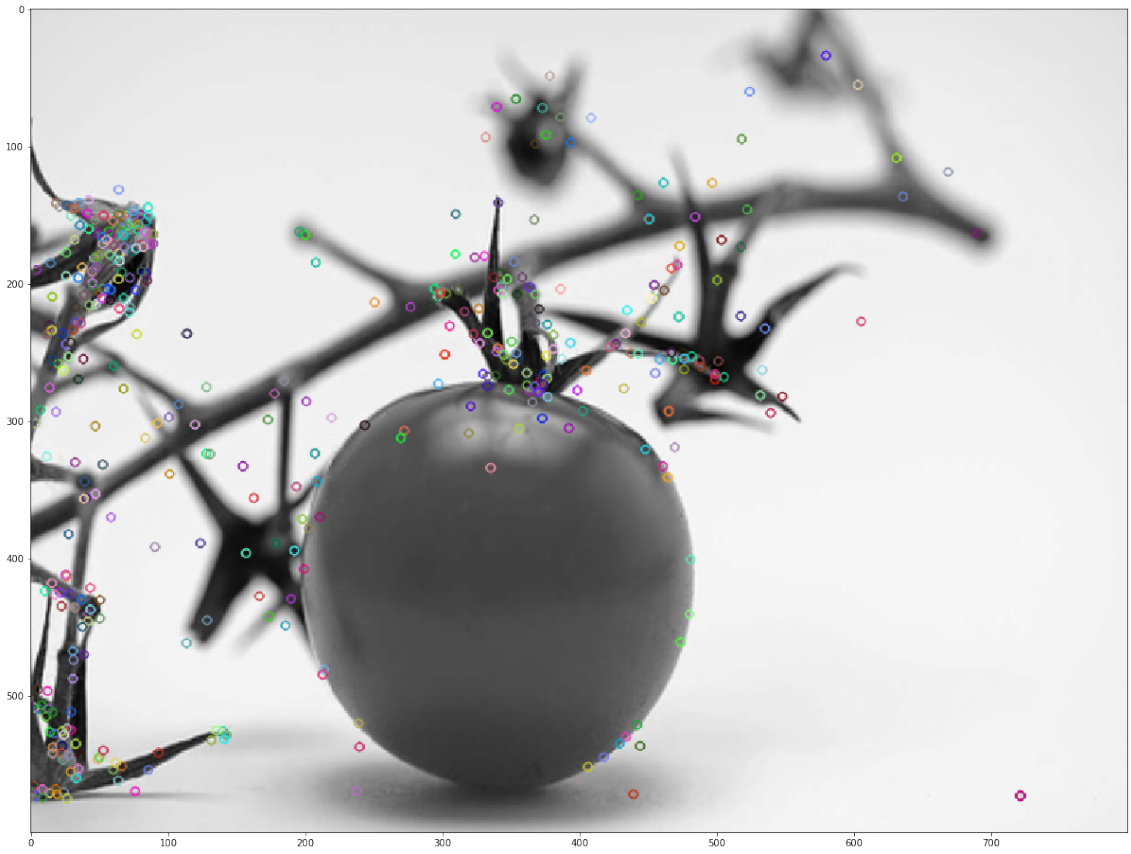


Figure 2: Detected features using SURF.

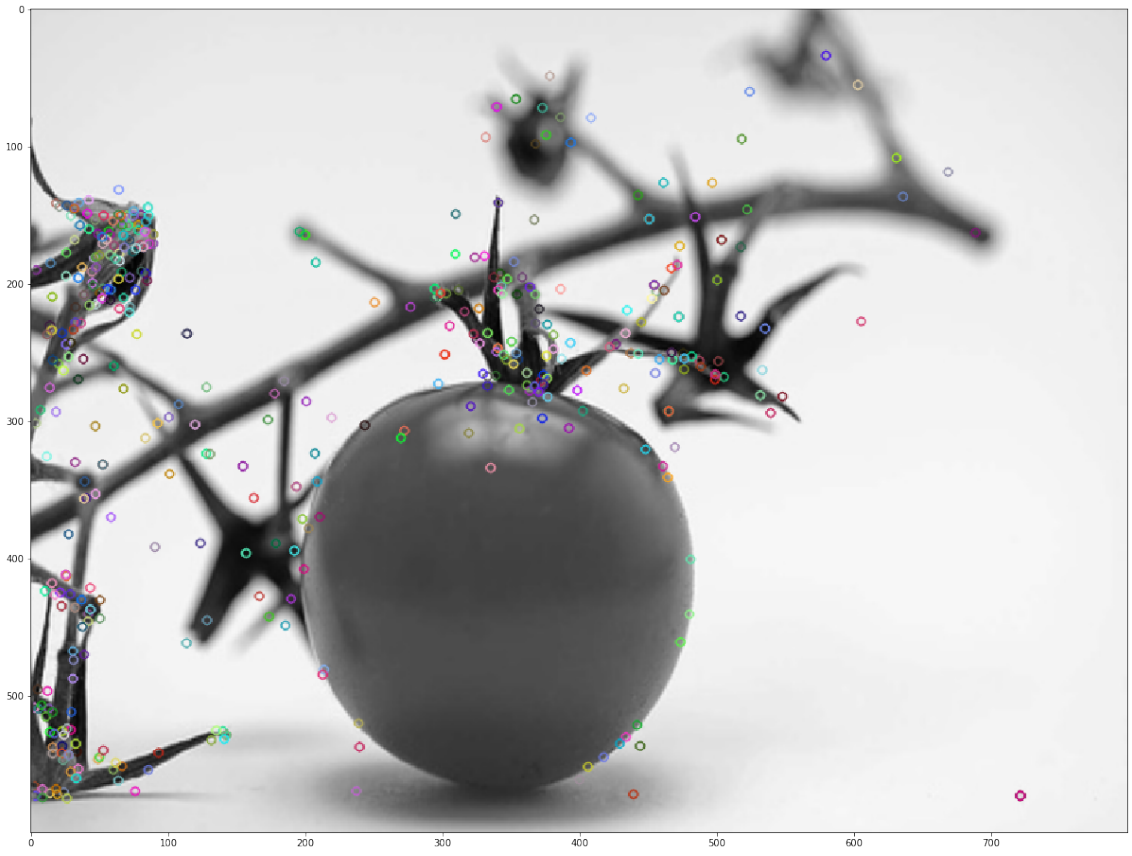


Figure 3: Detected features using ORB.

List of Figures

2.1 Graphical representation of an *McCulloch-and-Pitts-Neuron* 3
2.2 Visualization of a perceptron. 4
2.3 Example of a two-dimensional convolution 5
2.4 Examples of the different activation functions. 6
(a) ReLU 6
(b) Sigmoid 6
(c) Leaky ReLU 6
2.5 Example of *Max Pooling* and *Average Pooling*. Adapted from [38]. 7

3.1 Detected features using SIFT, SURF and ORB 12
(a) SIFT 12
(b) SURF 12
(c) ORB 12
3.2 Example of the watershed algorithm. 13
(a) Original image. 13
(b) Image converted into a two-color image. 13
(c) Segmented image using the watershed algorithm. 13

4.1 Architecture of VGG-16 16
4.2 Graphical representation of the Inception modul with demension reduction. 17
4.3 Representation of the new architecture of the Inception module used in Inception V2. 18
4.4 Illustration of the depthwise separable convolution used in Mobilenet V1 . . 19
(a) Convolution using a 3×3 kernel. 19
(b) Depthwise separable convolution used in Mobilenet V1. 19
4.5 Comparisation of the architectures used in MobileNet V1 and MobileNet V2. 20
(a) Illustration of the depthwise separable convolution layers used in MobileNet V1. 20
(b) Architecture of the bottleneck residual blocks used in MobileNet V2 20
4.6 Example of the anchor boxes used in Faster RCNN 22

4.7 Architecture of *Faster R-CNN*. 23
4.8 Architecture of R-FCN. 24
4.9 Overview of SSD 25
4.10 YOLOv3 architecture 26

5.1 Number of objects per class in the training dataset 29
5.2 Example images from the three evaluation data sets. 31
(a) First data set 31
(b) Second data set 31
(c) Third data set 31
5.3 Images of the prototype used in the context of this thesis. 32
(a) Image of the Raspberry Pi 4 B used in this thesis 32
(b) Image of the Raspberry Pi camera module 32

6.1 Graphical representation of the calculation of the IoU. 36
6.2 Examples of Precision × Recall curves. 36
(a) Precision × Recall of the class beerbottle. 36
(b) Precision × Recall of the class tomato. 36

7.1 Example images showing detection results using Faster R-CNN Inception V2 48
(a) Faster R-CNN Inception V2 (Non modified version) 48
(b) Faster R-CNN Inception V2 (Frozen feature extractor layer) 48
7.2 Example of the background subtraction. 57
(a) Image I_{t_0}, which represents the initial status. 57
(b) Image I_{t_1}, where a mini banana was added. 57
(c) Visualization of the background subtraction. 57
7.3 Illustration of the developed pipeline, which combines the background subtraction with an object detector. 57
7.4 Speed measurements of the pipeline using different input sizes. 61

1 Detected features using SIFT. 71
2 Detected features using SURF. 72
3 Detected features using ORB. 73

Bibliography

[1] Coco Detection Challenge. https://competitions.codalab.org/competitions/5181. Last accessed: February 17, 2020.

[2] Open Images Challenge 2019. https://storage.googleapis.com/openimages/web/challenge2019.html. Last accessed: February 17, 2020.

[3] The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Development Kit. http://host.robots.ox.ac.uk/pascal/VOC/voc2012/htmldoc/devkit_doc.html#SECTION00050000000000000000. Last accessed: February 17, 2020.

[4] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I. J., Harp, A., Irving, G., Isard, M., Jia, Y., Józefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D. G., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P. A., Vanhoucke, V., Vasudevan, V., Viégas, F. B., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *ArXiv abs/1603.04467* (2015).

[5] Alvarez, D., Cerezo-Hernández, A., López-Muñiz, G., Castro, T., Albi, T., Hornero, R., and del Campo, F. *Usefulness of Artificial Neural Networks in the Diagnosis and Treatment of Sleep Apnea-Hypopnea Syndrome*. 04 2017, pp. 33–68.

[6] Bay, H., Tuytelaars, T., and Van Gool, L. Surf: Speeded up robust features. In *Computer Vision – ECCV 2006* (Berlin, Heidelberg, 2006), A. Leonardis, H. Bischof, and A. Pinz, Eds., Springer Berlin Heidelberg, pp. 404–417.

[7] Beucher, S., and Lantuéjoul, C. Use of watersheds in contour detection. vol. 132.

[8] Calonder, M., Lepetit, V., Strecha, C., and Fua, P. Brief: Binary robust independent elementary features. In *Proceedings of the 11th European Conference on Computer Vision: Part IV* (Berlin, Heidelberg, 2010), ECCV'10, Springer-Verlag, pp. 778–792.

[9] Cartucho, J. Github: Cartucho/mAP. `https://github.com/Cartucho/mAP`, May 2019. Last accessed: February 17, 2020.

[10] Chakraverty, S., Sahoo, D. M., and Mahato, N. R. *McCulloch–Pitts Neural Network Model*. Springer Singapore, Singapore, 2019, pp. 167–173.

[11] Cun, Y. L., Boser, B., Denker, J. S., Howard, R. E., Habbard, W., Jackel, L. D., and Henderson, D. *Handwritten Digit Recognition with a Back-Propagation Network*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990, p. 396–404.

[12] Dai, J., Li, Y., He, K., and Sun, J. R-fcn: Object detection via region-based fully convolutional networks, 2016.

[13] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09* (2009).

[14] Everingham, M., Eslami, S. M. A., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision 111*, 1 (Jan. 2015), 98–136.

[15] Everingham, M., Gool, L. V., Williams, C. K. I., Winn, J., and Zisserman, A. The pascal visual object classes (voc) challenge.

[16] Ferguson, M., ak, R., Lee, Y.-T., and Law, K. Automatic localization of casting defects with convolutional neural networks. pp. 1726–1735.

[17] Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics 36*, 4 (Apr 1980), 193–202.

[18] Girshick, R. Fast r-cnn. *2015 IEEE International Conference on Computer Vision (ICCV)* (Dec 2015).

[19] Girshick, R., Donahue, J., Darrell, T., and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation, 2013.

[20] Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. The MIT Press, 2016.

[21] Hahnloser, R. H., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., and Seung, H. S. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature 405*, 6789 (2000), 947.

[22] Harris, C., and Stephens, M. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference* (1988), pp. 147–151.

[23] Hassaballah, M., Abdelmgeid, A. A., and Alshazly, H. A. Image features detection, description and matching.

[24] Hollemans, M. Google's mobilenets on the iphone. =https://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/, Jun 2017. Last accessed: February 17, 2020.

[25] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[26] Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., and Murphy, K. Speed/accuracy trade-offs for modern convolutional object detectors.

[27] Huttenlocher, D. P., and Ullman, S. Recognizing solid objects by alignment with an image. *Int. J. Comput. Vision 5*, 2 (Nov. 1990), 195–212.

[28] Ioffe, S., and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[29] Karlik, B., and Olgac, A. V. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems 1*, 4 (2011), 111–122.

[30] Kushibar, K., Valverde, S., González-Villà, S., Bernal, J., Cabezas, M., Oliver, A., and Llado, X. Supervised domain adaptation for automatic sub-cortical brain structure segmentation with minimal user interaction. *Scientific Reports 9* (12 2019).

[31] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. *Lecture Notes in Computer Science* (2014), 740–755.

[32] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science* (2016), 21–37.

[33] Lowe, D. G. *Perceptual Organization and Visual Recognition.* Kluwer Academic Publishers, Norwell, MA, USA, 1985.

[34] Lowe, D. G. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision 60*, 2 (Nov 2004), 91–110.

[35] Maas, A. L., Hannun, A. Y., and Ng, A. Y. Rectifier nonlinearities improve neural network acoustic models.

[36] Mcculloch, W., and Pitts, W. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics 5* (1943), 127–147.

[37] Nair, V., and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning* (Madison, WI, USA, 2010), ICML'10, Omnipress, p. 807–814.

[38] Rawat, W., and Wang, Z. Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation 29* (06 2017), 1–98.

[39] Redmon, J. Darknet: Open source neural networks in c. `http://pjreddie.com/darknet/`, 2013–2016. Last accessed: February 17, 2020.

[40] Redmon, J., and Farhadi, A. Yolov3: An incremental improvement. *arXiv* (2018).

[41] Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015.

[42] Roberts, L. G. *Machine perception of three-dimensional solids.* PhD thesis, Massachusetts Institute of Technology, 1963.

[43] Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review 65*, 6 (1958), 386–408.

[44] Rosten, E., and Drummond, T. Machine learning for high-speed corner detection. In *ECCV* (2006).

[45] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. *Learning Representations by Back-Propagating Errors.* MIT Press, Cambridge, MA, USA, 1988, p. 696–699.

[46] Salman, S., and Liu, X. Overfitting mechanism and avoidance in deep neural networks, 2019.

[47] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (Jun 2018).

[48] Schmidt, T. G., Schneider, F., Leverenz, D., and Hafner, G. *Lebensmittelabfälle in Deutschland - Baseline 2015*, vol. 71 of *Thünen-Report.* vTI, Braunschweig, 2019.

[49] Simonyan, K., and Zisserman, A. Very deep convolutional networks for large-scale image recognition, 2014.

[50] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Jun 2015).

[51] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Jun 2016).

[52] Tavares, J., and Natal Jorge, R. *VipIMAGE 2017 (Proceedings of the VI ECCOMAS Thematic Conference on Computational Vision and Medical Image Processing Porto, Portugal, October 18-20, 2017)*. 01 2018.

[53] Tensorflow. Github/tensorflow model zoo. `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md`. Last accessed: February 17, 2020.

[54] Tensorflow. Github: Tensorflow object detection sample config. `https://github.com/tensorflow/models/tree/master/research/object_detection/samples/configs`, Nov 2019. Last accessed: February 17, 2020.

[55] Torrey, L., and Shavlik, J. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI Global, 2010, pp. 242–264.

[56] Uijlings, J. R. R., van de Sande, K. E. A., Gevers, T., and Smeulders, A. W. M. Selective search for object recognition. *International Journal of Computer Vision 104*, 2 (Sep 2013), 154–171.

[57] Zhao, Z.-Q., Zheng, P., Xu, S.-T., and Wu, X. Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems PP* (01 2019), 1–21.

[58] Zhou, Y.-T., and Chellappa, R. Computation of optical flow using a neural network. In *IEEE International Conference on Neural Networks* (1988), vol. 1998, pp. 71–78.

[59] Zhu, M. Recall, precision and average precision.

[60] Zisserman, A., Mundy, J., Forsyth, D., Liu, J., Pillow, N., Rothwell, C., and Utcke, S. Class-based grouping in perspective images. pp. 183–188. Pro-

ceedings of the 5th International Conference on Computer Vision ; Conference date: 20-06-1995 Through 23-06-1995.

Eidesstattliche Versicherung
(Affidavit)

Kukkel, Christopher
Name, Vorname
(Last name, first name)

140185
Matrikelnr.
(Enrollment number)

Ich versichere hiermit an Eides statt, dass ich die vorliegende ~~Bachelorarbeit~~/Masterarbeit* mit dem folgenden Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present ~~Bachelor's~~/Master's* thesis with the following title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution.

Titel der ~~Bachelor~~-/Masterarbeit*:
(Title of the ~~Bachelor's~~/ Master's* thesis):

Development of perceptible prototype for detection and classification of food inside refrigerators

*Nichtzutreffendes bitte streichen
(Please choose the appropriate)

Dortmund, 17.02.2020
Ort, Datum
(Place, date)

Unterschrift
(Signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird gfls. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to €50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, section 63, subsection 5 of the North Rhine-Westphalia Higher Education Act (*Hochschulgesetz*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:**

Dortmund, 17.02.2020
Ort, Datum
(Place, date)

Unterschrift
(Signature)

**Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.