Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Detecting objects in images with known scene geometry

Bc. Matěj Suchánek

Supervisor: Ing. Jan Čech, Ph.D.
May 2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení:	**Suchánek**	Jméno: **Matěj**	Osobní číslo: **474573**
Fakulta/ústav:	**Fakulta elektrotechnická**		
Zadávající katedra/ústav:	**Katedra počítačů**		
Studijní program:	**Otevřená informatika**		
Specializace:	**Datové vědy**		

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Detekce objektů v obraze se známou geometrií scény

Název diplomové práce anglicky:

Detecting objects in images with known scene geometry

Pokyny pro vypracování:

Contemporary visual object detectors have achieved a great progress recently. The detectors are able to find many object categories while being real time. The detectors are general and do not take into account additional information on the objects and the scene geometry. A typical situation occurs when objects of known size are located on a given plane in the scene, e.g., a scenario of traffic cones in front of an autonomous student formula. In such a case, it is impossible that small objects were detected close to the camera, large objects at distance, or above the horizon.
Explore the problem and propose a way to incorporate the geometric information into the detection algorithm. Consider modifying a standard algorithm, e.g. [1,2], or postprocessing their output, or proposing a novel algorithm. Evaluate both the accuracy and computational time.

Seznam doporučené literatury:

1. S. Ren, K. He, R. Girshick, J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In NeurIPS, 2015.
2. J. Redmon, S. Divvala, R. Girshick, A. Farhadi. You only look once: Unified, real-time object detection. In CVPR, 2016.
3. A. Bulat and G. Tzimiropoulos. How far are we from solving the 2D & 3D Face Alignment problem? (and a dataset of 230,000 3D facial landmarks). In ICCV, 2017.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Jan Čech, Ph.D. skupina vizuálního rozpoznávání FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **08.02.2022** Termín odevzdání diplomové práce: **20.05.2022**

Platnost zadání diplomové práce: **30.09.2023**

Ing. Jan Čech, Ph.D.	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce		podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.

Datum převzetí zadání Podpis studenta

Acknowledgements

I would like to thank my supervisor Jan Čech for his invaluable continuous guidance. I would also like to thank the eForce FEE Prague Formula team for sharing their data from an experiment and their member Roman Šíp for sharing parts of his thesis on a related topic.

Declaration

Abstract

Recently, there has been great progress in object detection accuracy and speed. In autonomous driving, fast and accurate predictions are crucial. Current state-of-the-art methods do not constrain the detection according to the scene geometry. In the thesis, we study approaches to object detection based on deep learning, such as Faster R-CNN and YOLO, and propose a formal model of prior knowledge about the environment and its scene geometry. We assume that the objects have known size, are on the ground, and the camera parameters are known. On the basis of that, we propose three ways of including the prior knowledge in the algorithms. First, we modify the YOLOv3 detector by varying the dimensions of its expected bounding boxes. Second, we process the detections provided by a detector and ignore those with impossible dimensions. Third, we develop a heatmap regression model for object detection that relies on the prior knowledge. We experiment with the prototypes in cone detection for an autonomous student formula. We observe a slight improvement in isolated cases. We conclude that deep neural networks learn the constraints implicitly from the training data and that our proposals do not have much space for improvement.

Keywords: object detection, computer vision, YOLO, neural networks, deep learning

Supervisor: Ing. Jan Čech, Ph.D.

Abstrakt

V poslední době se podařilo značně zrychlit a zpřesnit detekci objektů. Rychlou a přesnou detekci vyžadují autonomní vozidla. Soudobé moderní metody však detekci s ohledem na geometrii scény nijak neomezují. V této práci se zabýváme postupy detekce objektů založenými na hlubokém učení, jako jsou Faster R-CNN a YOLO, a navrhujeme formální model povědomí o prostředí a geometrii jeho scény. Předpokládáme, že objekty mají známou velikost, nacházejí se na zemi a parametry kamery jsou taktéž známé. Na základě toho navrhujeme tři způsoby inkluze povědomí o prostředí do algoritmů. Za prvé, úpravu detektoru YOLOv3, aby se lišily rozměry očekávaných bounding boxů. Za druhé, kontrolu detekcí zjištěných detektorem a vylučování těch s neodpovídajícími rozměry. Za třetí, realizaci detekce pomocí modelu pro výpočet map intenzity, který využívá povědomí o prostředí. S prototypy experimentujeme při detekci kuželů pro samořiditelnou studentskou formuli. Pozorujeme malé zlepšení v určitých izolovaných případech. Shledáváme, že hluboké neuronové sítě se učí omezení implicitně z trénovacích dat a našim návrhům již nenechávají moc prostoru pro zlepšení.

Klíčová slova: detekce objektů, počítačové vidění, YOLO, neuronové sítě, hluboké učení

Překlad názvu: Detekce objektů v obraze se známou geometrií scény

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Motivation

In object detection, we research an algorithm (called *detector*) that receives digital images as input and is supposed to find some objects of interest in them. As a response, it tells what their locations are within the image and what kind of an object they are (an apple, a car, a human, etc.). The image may depict no objects or multiple objects of the same or different kinds.

Algorithms of object detection find application, for example, in the nowadays very popular self-driving car industry. A self-driving car needs to quickly and reliably determine the position (distance) of various objects (traffic signs, other vehicles, pedestrians, etc.) in front of it to drive safely.

The application range of solutions to object detection is relatively broad, and there is naturally a trade-off between the variety of supported applications and their performance in specific applications. We notice that state-of-the-art approaches are general and do not provide any means of further adaptation to the surrounding environment. For example, in the environment of a self-driving vehicle, pedestrians will appear taller than wider, unlike other vehicles around, and they will stay on the ground, unlike traffic signs.

Suppose it was possible to provide this prior knowledge about the environment to the detectors, e.g., by applying some environment-specific constraints based on the scene geometry. In that case, it could help to improve the detector's performance. For example, we could make an algorithm simply ignore detections that do not satisfy the knowledge model. Ultimately, this may improve its robustness and allow it to increase its greediness, so overlooking potentially important objects is less likely.

In this thesis, we make a review of state-of-the-art methods and propose an appropriate model for the prior knowledge. Based on these findings, we formulate several proposals for improving existing algorithms and do experiments involving them.

1.2 Thesis outline

We divide the thesis into three main parts. In the first part (Chapter 2), we formulate the object detection problem and review existing state-of-the-art approaches by describing their architecture and features. We also examine algorithms used in other related fields of computer vision and identify ideas that may be carried over and used to improve general algorithms for object detection.

In the second part (Chapter 3), we describe some familiar computer vision essentials and present suggestions on how they could be combined with existing object detection algorithms to improve their performance. In the last part (Chapter 4), we present the results of our experiments with the proposed enhancements and compare their performance with baseline state-of-the-art methods.

Chapter 2

Related work review

Like in other problems in computer vision, or machine learning in general, state-of-the-art solutions to object detection are based on deep learning. We describe an early face detector (Sec. 2.2) and the development and design of two popular "families" of ANN architectures, R-CNNs (Sec. 2.3) and YOLO (Sec. 2.4). We also summarize ideas from other related research fields that we later adopt and experiment with (Sec. 2.5).

First, though, we present our general overview of object detection as a process, as well as the concepts related to it, and some practical remarks. In particular, R-CNNs and YOLO follow this schema.

2.1 Overview of object detection paradigm

We stick to the following object detection paradigm and terms throughout the thesis.

Objects of interest are located in the surrounding 3D world (scene). An agent interested in detecting objects carries a camera taking 2D images of the scene and uses an algorithm to perform object detection. Images are represented as a multidimensional array organized by image pixels.

The algorithm takes one or more multiple images as the input and produces a certain number (zero, one, or more) of detections for each one. Each detection holds information about its location within the image. The location is represented as a rectangle (called "bounding box"), which circumscribes the detected object in the image. Its parameters can be expressed in multiple equivalent ways, for example:

- its width and height (in pixels) and coordinates (offset in each dimension in pixels) of its top-left corner,
- its width and height (in pixels) and coordinates of its center,
- coordinates of its top-left and bottom-right corner,

etc. The next information about a detection is one of the categories that the agent knows and wants to distinguish from each other.

Finally, the algorithm may assign each detection some kind of “score” or confidence value, usually between 0 and 1 (the higher the value, the more confident the algorithm is about the object’s presence). Prior to inference, the agent chooses a threshold and only considers detections with a score higher than the threshold as actual detections. Determining the best value for the threshold is a manifestation of a common dilemma of machine learning algorithms – there is a trade-off between the number of false-positive (“invalid”) detections and the number of false-negative (“missed”) detections.

The choice of threshold determines how sensitive the algorithm is supposed to be, that is, whether and how much it prefers *precision* over *recall* (sensitivity, “greediness”) or vice versa.[1] While the preference for the recall may prevent the algorithm from overlooking potentially important objects, it will also likely increase the number of invalid detections, which may cause the agent’s confusion (e.g., when planning the path).

2.2 Viola–Jones object detector

Prior to deep learning, one of the earliest detectors was the Viola–Jones object detection framework. It was primarily focused on face detection, that is, finding a human face in the image. The detector would learn to classify features in a data structure called *integral image* using the AdaBoost classifier. It achieved real-time performance [3].

2.3 R-CNNs

R-CNN (Region Based Convolutional Neural Networks) is an early family of deep neural networks for object detection. The initial version of R-CNN uses selective search algorithms to extract regions of interest (RoI) from the input image. Each region is then evaluated by a convolutional neural network (CNN), returning feature maps that are classified using support vector machines (SVM). This approach was inefficient, as it took tens of seconds to process an image [4, 5].

The next version, Fast R-CNN, changed the order of operations. It computes the feature maps for the whole image only once and identifies RoIs in them. It still uses selective search to identify RoIs, but it replaced SVM classifiers with the standard classification module. Classification is carried out on a fixed-size feature vector which is extracted from the feature map for each proposed RoI [6, 5].

Its successor, Faster R-CNN, replaced the selective search algorithm for proposing regions of interest with a region proposal network (RPN), whereby it became fully end-to-end trainable and achieved real-time performance. Internally, it still involves a loop over the proposals. The RPN also exploits the concept of *anchor boxes* (called “anchors”), which was also adopted by YOLO [7, 5].

[1]Formal definition of these two statistics is provided in Sec. 4.2.

2.4 YOLO

As the name "You Only Look Once" suggests, the YOLO architecture evaluates the input image (and its feature map) *only once*. It is categorized as a *single-stage detector*, unlike Faster R-CNN (which is classified as a *two-stage detector*). The architecture has been continually improved since it was first published [8, 1, 9]. We focus on and describe the third edition – the YOLOv3 architecture.

2.4.1 YOLOv3

The YOLOv3 network is divided into two modules. The first is a "backbone" deep fully-convolutional network (called DarkNet). The second one is a "tail" module which behaves as a feature pyramid network (FPN) and has three levels. Each level generates detections from features with different scale (resolution). The ultimate output of YOLOv3 is a union of these detections. The FPN is supposed to help YOLOv3 detect objects of various sizes [9].

An important hyperparameter of YOLOv3 is a set of *anchor boxes* (also called *bounding box priors*). They represent the expected dimensions (width and height) of rectangular bounding boxes of detected objects. YOLOv3 allows for the choice of custom anchor boxes and assigns them to each output scale. Anchor boxes must be chosen before the training phase and cannot be changed afterwards because the weights are learned relative to anchor boxes.

YOLOv3 splits outputs from the FPN into units called "cells" and yields detections per cell and scale. Each cell corresponds to a portion of the input image and is responsible only for detections in that region. The regions do not overlap, and their shape is square [8, 1, 9]. The original implementation has three detections per cell and scale.

For each cell on every scale, YOLOv3 returns the following array of values [9]:

- the "objectness" score (probability that there is an object) $\in (0.0; 1.0)$
- offset of the location of the detection center relative to the top left corner of the cell and the dimensions of the cell $\in (0.0; 1.0)^2$
- natural logarithm of the quotient of anchor box width and height deviation $\in \mathbb{R}^2$
- C conditional class probabilities (given objectness) $\in (0.0; 1.0)^C$

During training, for each ground truth sample, the detection with the greatest overlap (measured using the Intersection over Union (IoU) metric) is chosen. It is treated as true positive and backpropagates objectness, coordinate (location and size), and classification errors. Other detections that have significant overlap with ground truth, but not the greatest, do not backpropagate. All other detections are treated as false positive and backpropagate objectness errors. During inference, only detections with an objectness score above a

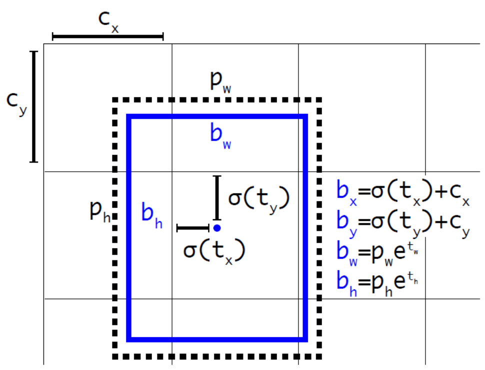


Figure 2.1: Visualization of how individual outputs from YOLOv3 determine the position of a detected object. The dashed line corresponds to the anchor box. The image was taken from [1].

threshold (e.g., 0.5) are considered. They may be further post-processed by the *non-max suppression algorithm* so that overlapping (likely duplicate) detections are removed [9].

2.5 Point-based detection and heatmap regression

The two presented methods share common properties – they directly predict the object location and size as a list of tuples of four values: offset from the border in each dimension and size in each dimension (i.e., width and height). They also use anchor boxes and let their predictions be relative to them.

We believe that having all these pieces of information is redundant if we already have enough knowledge about the scene geometry. We looked into recent solutions to other related problems in computer vision for ideas for an alternative approach.

During the research, we noticed a recurring proposal in several papers – formulating problems as *heatmap regression*. These include solutions to face alignment (facial landmarks localization) [10, 11], human pose estimation [12, 13], treatment planning [14] and also object detection problems – some very recent architectures of deep networks for object detection, like CornerNet [15] and ExtremeNet [16], use heatmaps instead of anchor boxes to predict positions of certain significant points (multiple per detected object).

These methods do not regress location directly (i.e., they do not predict the exact pixel offset), but infer it from a regressed *heatmap* as the output – a data structure that holds information about each pixel of the input (i.e., maps each pixel to the information). The information is usually some intensity measure that increases towards the keypoint's location (pixel), and it is often constructed from a Gaussian kernel.

Heatmap regression is similar to the semantic segmentation problem, where the expected output is classification of all input image's pixels (pixel-to-class map). It is usually formulated as a pixel-wise regression of class logits (probabilities). Semantic segmentation was designed for processing medical images or surface segmentation. A notable architecture of deep neural networks for semantic segmentation is U-Net [17].

These findings make us believe it is possible to narrow our task to predict the location of an only significant point (*keypoint*) and reconstruct the object's most likely position and dimensions from it using knowledge about the scene geometry. We later introduce means of how it can be done in practice (Sec. 3.1) and propose a modification of existing architecture for purposes of object detection that borrows some of these ideas (Sec. 3.4).

Chapter 3

Proposed methods

In this chapter, we present three different methods that we believe can improve the accuracy of existing object detection algorithms. They all utilize prior knowledge about the scene and its geometry in various ways and thus are only applicable if the knowledge is available.

The first section of this chapter summarizes what kind of knowledge is useful and necessary.

3.1 Prior knowledge representation

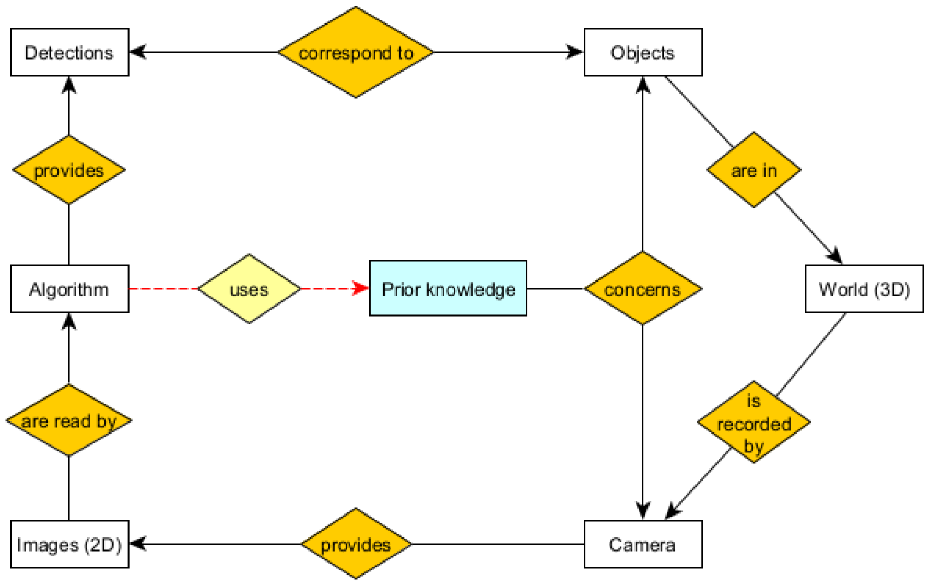


Figure 3.1: Visualization of relationships between concepts related to object detection. Our goal is to exploit the potential relationship between "Algorithm" and "Prior knowledge" (red arrow). The figure is our own work generated using the yEd editor.

In the beginning, we looked at what *prior knowledge* can actually concern and how it can be represented. Our mental model, which is also based on our overview of object detection outlined in Sec. 2.1, is presented in Fig. 3.1.

First, the prior knowledge may concern the *geometry of the objects of interest*. Complete representation of their geometry is often complex, and hence not suitable. Since the produced detections (and also the ground truth) are shaped as rectangles, we believe it is sufficient to represent the objects' geometry as a tuple of dimensions of the respective cuboids, i.e., their width, depth, and height. Knowledge about the expected position (exact or relative to the agent) of the objects (e.g., on the ground, in the distance, etc.) may also be involved.

The second aspect is the *camera*, or the process of taking images in general. Knowledge about the objects naturally applies to the 3D world. However, the algorithms for object detection read the (2D) images of the world where knowledge about objects' dimensions does not apply directly. Obviously, it is necessary to be aware of the process by which the images of the scene are produced, and it is advantageous if it can also be simulated. We call this knowledge the "camera model" and describe its representation in the following section.

3.1.1 Camera model

We build our prior knowledge model about the camera on computer vision essentials and use the *pinhole camera model* [18]. It allows us to simulate the environment (world) and control the mapping from the 3D *world coordinates* to the 2D coordinates on the projective plane (called *pixel coordinates*) using linear algebra [19].

The relation between the world and pixel coordinates of a point can be expressed by the following equation [19, 20]:

$$z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{T} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{3.1}$$

The parameters u, v represent the pixel coordinates, x_w, y_w, z_w represent the world coordinates. The two vectors correspond to the two points' *homogeneous coordinates* [21] – they have an additional dimension with the value 1. z_c is a real number and is chosen so that the equation is satisfied.

$\mathbf{K}$ is the 3×3 intrinsic camera matrix. It holds the camera's intrinsic parameters, such as the focal length. $\mathbf{R}, \mathbf{T}$ are the 3×3 rotation matrix and the 3×1 translation vector, respectively. These specify the transformations needed to map the scene to the camera-centered coordinates. $\mathbf{K}$, $\mathbf{R}$ and $\mathbf{T}$ are usually estimated using computer vision algorithms (not the subject of this research) [19, 20].

Given an object in the world whose dimensions (i.e., width, depth, and height) and location are known, we can now determine its position on the projective plane (image) of our camera. However, this mapping is not reversible – multiple points in the world's 3D space can be projected to a single point in the 2D image space [19]. ("A single point can be represented by infinitely many homogeneous coordinates." [21])

3.2 Variable anchor size in YOLOv3

The concept of *anchor boxes* (explained in Sec. 2.4) is, in fact, a kind of prior knowledge. If we know how large the objects we want to detect are and how large they will appear in the image, we may choose anchor boxes with dimensions close to the expected object dimensions. It was proposed that instead of arbitrarily choosing their dimensions, they can be determined by running the k-means algorithm on a large training dataset, such as COCO [1, 22].

However, we also need to keep in mind the following: *distant objects appear smaller than nearby objects of the same size.* This principle is particularly important in the environment in which we would like to apply our research – the agent is an autonomous formula with a camera, and the objects to be detected are traffic cones on the road. YOLOv3 anchor boxes, however, use fixed dimensions for all regions of the image [9].

Thus, we want to explore the possibility of modifying the YOLOv3 detector to use anchor boxes of dimensions that would vary for different regions of the image depending on how large the objects will appear in them. If we could determine that for each region (YOLOv3 organizes the input into cells and assigns the same number of anchor boxes to each [9]), we would be able to precompute the anchor box sizes, train the model relative to them, and use them during inference in the same environment.

Notice, though, that this procedure starts from the opposite end – we know the position in the image and want to determine the location in the world. The usual process, i.e., taking images, goes from 3D (scene, objects) to 2D (image), and we already know it is not reversible by default (Sec. 3.1.1). The latter fact is manifested by the parameter z_c in the mapping, eq. (3.1), which represents the degree of freedom even if the parameters u and v (pixel coordinates of the YOLOv3 cell / anchor box center) are fixed.

Here, we will exploit the prior knowledge about the scene and its geometry. In particular, we know that traffic cones stay on the ground in our environment and we also know their dimensions. This information, together with the complete camera model (Sec. 3.1.1), is sufficient – we know the location and orientation of the camera and can infer the z-coordinate of a point based on the orientation of the ground plane (where usually $z = 0$). Fixing the z_w parameter no longer leaves an extra degree of freedom.

Formally, we want to express x_w and y_w as a function of u, v, and z_w, given the camera model. It can be derived as follows:

First, we rewrite the eq. (3.1) and then multiply both sides by the inverse matrices of **K** and **R** from the left. **R** is a rotation matrix; hence, its inverse is its transposition:

$$\begin{aligned}
z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{T} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}, \\
z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \mathbf{K}(\mathbf{R} \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + \mathbf{T}), \\
z_c \mathbf{K}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \mathbf{R} \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + \mathbf{T}, \\
z_c \mathbf{R}^T \mathbf{K}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + \mathbf{R}^T \mathbf{T}.
\end{aligned}$$

Then, since $u, v, \mathbf{K}, \mathbf{R}, \mathbf{T}$ are known, we make the following two assignments:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} := \mathbf{R}^T \mathbf{K}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}, \qquad \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix} := \mathbf{R}^T \mathbf{T}$$

and substitute them in:

$$z_c \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} x_w + r_1 \\ y_w + r_2 \\ z_w + r_3 \end{bmatrix}.$$

This is a system of three equations, where z_c, x_w and y_w are the variables (though we are only interested in x_w and y_w). We express z_c from the third equation and x_w, y_w from the other two:

$$\begin{aligned}
z_c a_3 &= z_w + r_3 & z_c a_1 &= x_w + r_1 & z_c a_2 &= y_w + r_2 \\
z_c &= \frac{z_w + r_3}{a_3} & x_w &= z_c a_1 - r_1 & y_w &= z_c a_2 - r_2
\end{aligned}$$

and by substituting the expression for z_c into them, we get the result:

$$x_w = \frac{a_1(z_w + r_3)}{a_3} - r_1 \tag{3.2}$$

$$y_w = \frac{a_2(z_w + r_3)}{a_3} - r_2. \tag{3.3}$$

Now that we can also determine complete world coordinates based on the pixel coordinates, camera model, and knowledge about the scene geometry, we can precompute all the anchor box dimensions. We propose the following "ray-casting" algorithm for computing an object's bounding box given its central point in the pixel coordinates:

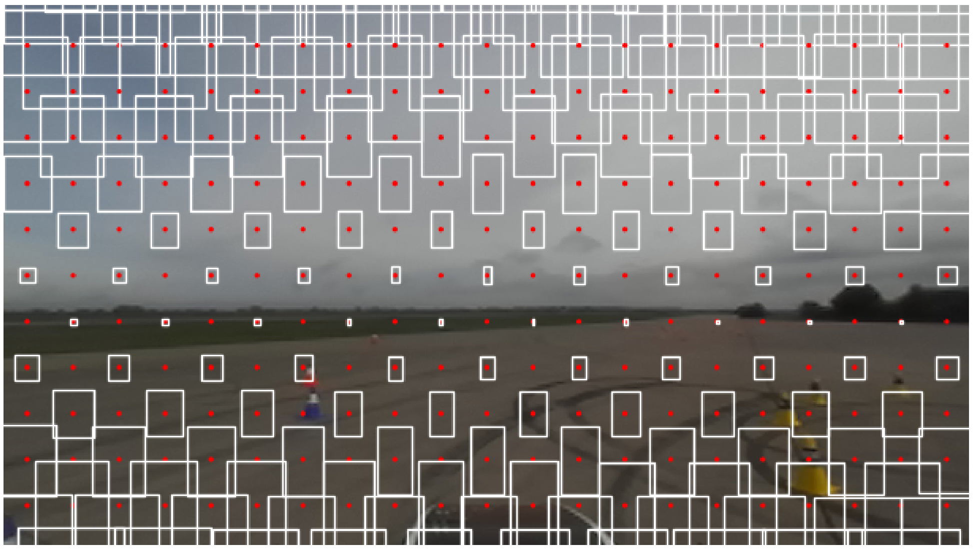


Figure 3.2: Visualization of a layer of anchor boxes with variable size. The red dots are cell centers (stride = 32) and the white rectangles are the corresponding anchor boxes. For better clarity, only every other anchor box is displayed.

1. Cast an imaginary ray (line) from the camera center through the cell center on the projective plane.
2. Record world coordinates (i.e., x_w and y_w) of a point on the ray whose z_w-coordinate is equal to the expected z_w-coordinate of the center of the object (substitute in equations (3.2) and (3.3)). More precisely, this is also the central point of the smallest rectangular cuboid spanning the object.
3. Determine coordinates of the eight vertices of the imaginary cuboid using knowledge about the expected object's dimensions.
4. Project the eight vertices back into the projective plane (substitute x_w, y_w, and z_w in eq. (3.1) and solve for u and v). This step can be vectorized by arranging the vectors of world coordinates (right-hand side of eq. (3.1)) into a matrix.
5. Find the smallest spanning rectangle of the vertices on the projective plane by computing the minimal and maximal coordinates in both dimensions. The width and height of the smallest spanning rectangle are the dimensions of the desired bounding box.

Now, we run the above algorithm for all YOLOv3 cells' centers. We do it for each output layer from the feature pyramid network (FPN) separately, as the cells on each level are organized differently. The example result of the algorithm is visualized in Fig. 3.2. Notice that anchor boxes were also computed for the upper part of the image, which captures the sky above the horizon. However, the objects are expected to be located below the horizontal line, close to the ground. In fact, these are projections of (imaginary) objects that are located *behind* the camera. We consider these anchor boxes harmless, since no traffic cones are expected to be detected above the horizon. (We propose using this knowledge during post-processing, see further.)

We note that models of the YOLOv3 architecture modified in this way may lose the ability to accept images of variable size, which fully-convolutional networks usually have.

Note that the algorithm is more general – the pixel coordinates do not necessarily have to correspond to a YOLOv3 cell's center but to any point in the image. Therefore, we also make use of it in the other proposed methods that we introduce later.

3.3 Filtering detections using camera model

This proposal focuses directly on the problem of increasing the rate of false-positive (invalid) detections with increasing the algorithm's sensitivity. It exploits the model of the environment based on our knowledge about the camera and the scene geometry. Unlike our previous proposal (Sec. 3.2), it can be used to enhance an arbitrary detector.

We propose running the detector unmodified and just applying a filter to the detections it returns (it happens as a part of the *post-processing* phase). Formally, let D be the set of detections produced by the detector, and let $filter$ be a function that takes a detection from the set as the input and returns a boolean. The procedure (in pseudocode) is the following:

```
function FilterDetections($D$, $filter$)
    $D_{filter} \leftarrow \emptyset$
    for $det \in D$ do
        if $filter(det)$ then
            $D_{filter} \leftarrow D_{filter} \cup \{det\}$
        end if
    end for
    return $D_{filter}$
end function
```

The filters that we propose and explore are based on knowledge of the scene geometry (that is, the camera model and the dimensions of the objects).

3.3.1 Location-based filters

The first type of filter is based on location. For example, if the agent is an autonomous student formula that detects colored traffic cones on the ground, it can use a filter that removes detections located above the horizon. The filter then compares the v-coordinate of a representative point of the detection (e.g., its center-middle or center-bottom point) and the v-coordinate of the horizontal line. The position of the horizontal line can be inferred by projecting a point in infinity using eq. (3.1). In practice, this means setting y_w and z_w to zero and x_w to a large number and solving for u.[1]

3.3.2 Similarity-based filters

A more advanced type of filter is based on the similarity (or deviation) of properties of the "expected" detection and the predicted one. The expected detection is functionally dependent on the predicted one (given the scene geometry). We can determine the expected bounding box dimensions using the ray-casting algorithm (Sec. 3.2, page 13) – by casting the ray from the camera center through the center of the detection's rectangle (pixel coordinates u, v).

After that, we can determine the similarity of the expected and predicted detections by evaluating a certain similarity metric. For example, the *intersection over union* (IoU) measure is a well-known metric that is often used during detector accuracy evaluation. Detections that do not conform to the camera model can be removed by computing the IoU of the expectation and prediction (a value between 0 and 1) and comparing the value with a threshold. Detections with IoU below this threshold are not considered. We refer to this as a "strict" filter. The threshold needs to be determined prior to inference, e.g., based on the training data. It may be a constant value between 0 and 1, but it may also functionally depend on the prediction. For example, the threshold may be relaxed for more distant predictions due to possible imprecision in the estimated camera model, or may vary for different classes of objects.

Alternatively, the output of a similarity function may be used to adjust the detection score prior to thresholding (Sec. 2.1). When the detector assigns each detection a confidence value/score (e.g., YOLOv3 objectness value), we propose combining it with the similarity score using an operator (function) and doing thresholding on its output instead. We refer to this as a "soft" filter. The function should be non-increasing for decreasing values of similarity so that the ultimate score is lower for detections that are not in accordance with our knowledge about the scene geometry. We experimented with combining the two scores by computing their product, which satisfies this criterion.

3.4 Heatmap regression using segmentation model

Based on the related work review (Sec. 2.5), we want to explore reformulating the object detection problem as heatmap regression using deep learning. This approach is different from traditional state-of-the-art architectures (Sec. 2.3 and 2.4) that regress multiple parameters of objects' bounding boxes individually.

We present our approach based on the proposal divided into three subsequent steps.

[1] Or similarly if another orientation of the axes was chosen.

3.4.1 Ground truth representation

First, we assign each object of the ground truth to a single keypoint. The relative location of the keypoint is the same for all objects, and its choice may vary for different types of objects. Furthermore, the bounding box should be recoverable from the keypoints. For example, in our environment of an autonomous formula, we placed the keypoints on the ground in front of traffic cones. In particular, if we are provided with the usual 2D annotations that assign each object a rectangular bounding box, the keypoint's location is the central point of its bottom line (see Fig. 3.3).

Next, for every image, we produce a set of heatmaps, i.e., 2D arrays with input image dimensions. Since we need to discriminate different classes of objects, there are separate heatmaps for each class of detected objects. (There is no additional class for the "background".) Each keypoint is mapped to the respective heatmap as a Gaussian kernel that is centered at the keypoint (where it also reaches its maximum value). Note that we do not normalize it nor have it integrate to one (like bivariate normal distribution); we simply let its maximal value be 1.

Formally, let D be the set of all image keypoints organized as tuples of three values: two pixel coordinates and the class index (x, y, c). Also, let $D_c = \{(x, y) | (x, y, c) \in D\}$ be the set of coordinates of keypoints of the given class. Then, the formula for all pixels and channels of the heatmap can be expressed as:

$$H[c, x, y] = \begin{cases} \max_{\boldsymbol{\mu} \in D_c} G_{\boldsymbol{\mu}, \boldsymbol{\Sigma}}(x, y), & |D_c| > 0 \\ 0, & \text{otherwise} \end{cases}$$

where $G_{\boldsymbol{\mu}, \boldsymbol{\Sigma}}(\mathbf{x}) = \exp(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}(\mathbf{x} - \boldsymbol{\mu}))$ and $\boldsymbol{\Sigma}$, the covariance matrix, must be chosen arbitrarily. We do not provide any hints for the optimal choice; instead, we only describe our observations in the experimental part of the thesis, where we experimented with setting the covariance matrix to several integral multiples of the identity matrix.

3.4.2 Architecture

We adopt U-Net, a fully-convolutional network architecture originally developed for semantic segmentation of biomedical images [17]. It takes images of arbitrary size with three input channels and produces pixel-wise class logits.[2] Instead of computing the argmax for each pixel to obtain image segmentation, we treat the output as a set of heatmaps. The only modification to the architecture that we made is the addition of a ReLU layer ($y_i = \max(0, x_i)$) after the output for learning that focuses on the foreground (objects). We propose training this model with the usual gradient descent method by optimizing L2 loss.

[2]The original paper only indicates a two-channel output map. Modified versions allow for an arbitrary number of output channels.

3.4.3 Output processing

The final step involves extracting the actual detections (location, class, and confidence) from the predicted heatmap. First, we determine the location of all potential keypoints and then assign a bounding box to each one. The latter part involves the knowledge about the scene geometry.

Heatmap to keypoints

We invert the process of creating the ground truth heatmap from keypoints (Sec. 3.4.1) by looking for pixels with the highest intensity in their neighborhood – local maxima.

The algorithm we propose for finding local maxima uses a sliding window. It evaluates each pixel with a sliding window (a square subset of the image domain) and finds the one with the highest intensity. If it is found in the sliding window's center (i.e., the center is the argmax), there is a local maximum and the keypoint is found in the window's center. The intensity value of the keypoint can be considered as its score. Keypoints with a score lower than a threshold (potential "noise") may be ignored.

The size of the sliding window is a hyperparameter. It should preferably be an odd integer greater than 1 so that the center is clearly determined. Increasing the size of the window will make the algorithm evaluate greater portions of the heatmap but also suppress keypoints that are located very close to each other – likely the same object. (This may be an alternative to the non-max suppression algorithm that YOLOv3 and other detectors use to remove overlapping detections.)

Two evaluation strategies are possible for combining results from multiple heatmaps that are supposed to discriminate the classes of detected objects:

1. Local maxima are searched in each heatmap separately, and the set of all keypoints in the image is the union of keypoints from individual heatmaps (complexity: $K^2 \times C \times w \times h$, where K is the kernel size).

2. The heatmaps are first reduced to a single heatmap of maximal intensities per pixel ($H_{max}[x, y] = \max_c H[c, x, y]$), and the algorithm only looks for local maxima in this heatmap. A keypoint's class can be looked up from the heatmap argmax array.
   We believe that this strategy has two advantages: it reduces computations (the sliding window runs only once per image, complexity: $(C+K^2)\times w \times h$), and it may prevent attribution of multiple objects of different classes to a single keypoint, especially in a situation where the presence of an object also "activates" the other heatmaps slightly. An implementation of this algorithm is demonstrated in Fig. 3.4.

Keypoint to bounding box

Finally, we map each identified keypoint to a bounding box. The exact process depends on the prior choice of its relative position. Note that if the keypoints do not correspond to the central point of the object, we cannot use the ray-casting algorithm (Sec. 3.2, page 13) and provide the keypoint as input. Instead, we exploit the prior knowledge and determine the object center within the scene (in world coordinates) based on the known object's dimensions. Then we continue with step 3 of the algorithm and determine the dimensions of the desired bounding box.

For example, if the keypoints are located in front of the objects on the ground (as we proposed in Sec. 3.4.1), we first compute the world coordinates of the keypoint by substituting z_w (of the ground plane), u and v in eq. (3.2) and (3.3). Then, we determine the world coordinates of the object's center using knowledge about its dimensions. The bounding box can now be determined by running the remaining steps of the ray-casting algorithm, which computes its dimensions. In the end, we put the bounding box in the appropriate position relative to the keypoint (in our environment, the keypoint is in the middle of the bottom line).

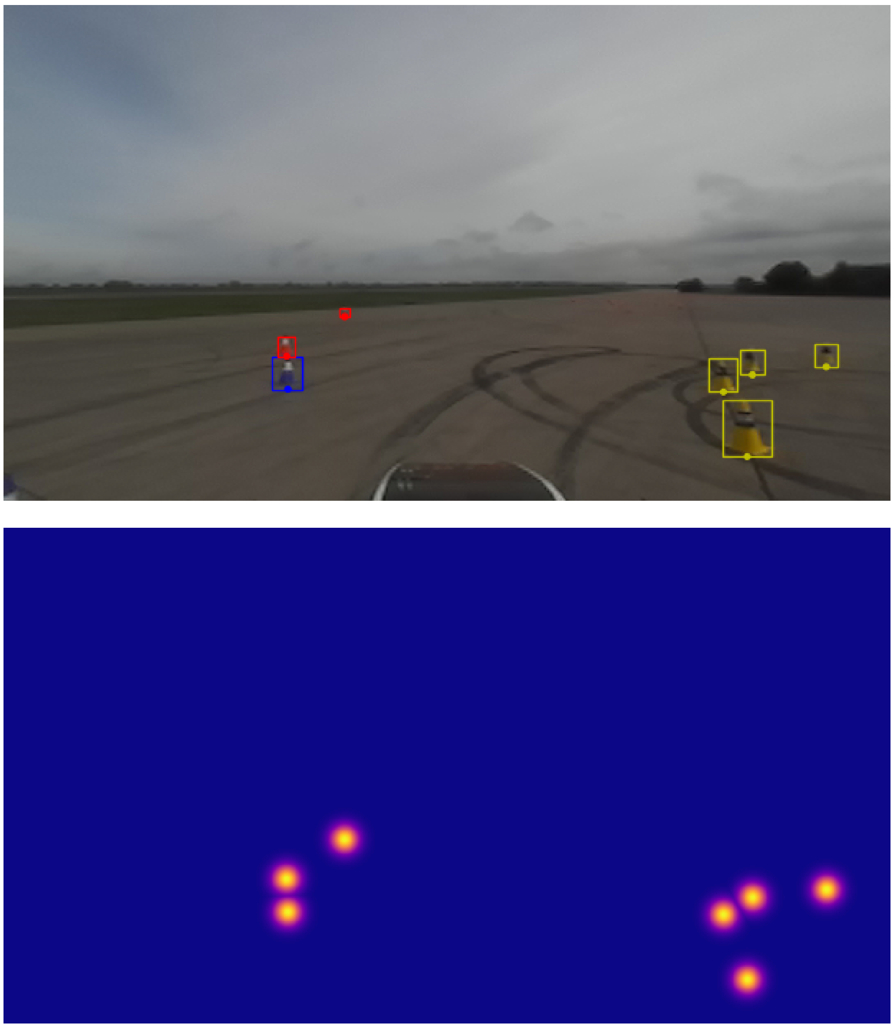


Figure 3.3: Example of the proposed representation of the ground truth for heatmap regression. The upper image shows a sample image, and objects annotated by the bounding boxes with the respective color and the keypoints (points on the bottom lines). The bottom image shows the corresponding heatmap (independent of classes; $\mathbf{\Sigma} = 50\mathbf{I}$).

```python
import torch.nn.functional as F

shape = images.size()
center_index = (kernel * kernel) // 2
prediction = model(images)
prediction = F.relu(prediction)
values, indices = prediction.max(1, keepdim=True)
sliding = F.unfold(
    values, kernel, padding=kernel // 2)
sliding = sliding.reshape(
    shape[0], kernel * kernel, *shape[2:])
maxima = sliding.argmax(1, keepdim=True) == center_index

nonzero = torch.nonzero(maxima)
for pos in nonzero:
    image_index = pos[0].item()
    keypoint_y = pos[2].item()
    keypoint_x = pos[3].item()
    score = values[pos].item()
    category = indices[pos].item()
```

Figure 3.4: Demonstration of the inference pipeline with the sliding window algorithm for finding local maxima (with heatmaps reduction) written in Python using the PyTorch library [2]. There are two parameters: `images` – the input to the network (tensor), and `kernel` – the size of the sliding window (integer).

Chapter 4
Experiments

In the last part of the thesis, we present the results of our experiments. We applied our proposals for using knowledge about the scene geometry and compared their results to uninformed methods based on state-of-the-art approaches.

4.1 Dataset

The dataset we used for the experiments consisted of a set of circa 670 images (376×672 pixels) taken by a camera on a self-driving student formula during an off-site experiment in Autumn 2019. They capture the scene in front of the formula, while it navigates a track delimited by traffic cones (the objects) in an airport.

The annotated objects have three categories: yellow, blue, and red traffic cones. We assumed that objects of all types were located on the ground and had equal diameter (i.e., width and depth) of 32 cm and height of 30 cm. Most importantly, we were also provided the parameters of the camera (Sec. 3.1.1).

4.2 Evaluation metrics

The key metrics we used for the comparison of competing methods were *average precision* (AP) and *mean average precision* (mAP) at IoU of 0.5. Predicted detections that have IoU with a ground truth detection at least 0.5 are considered true-positive (“hits”). Those for which no such overlap is found are considered false-positive (“invalid”), and ground truth detections for which no such overlap exists are considered false-negative (“missed”).

AP is defined as the area under the *precision-recall curve* (i.e., its integral). This curve represents a functional dependence of the precision on the recall. *Precision* is defined as the number of true-positive samples over the number of all positive samples ($\frac{TP}{TP+FP}$) and *recall* as the number of true-positive samples over the number of true-positive samples and false-negative samples ($\frac{TP}{TP+FN}$). The recall-precision pairs are sampled by changing the value for the threshold on the score of detections (confidence, objectness, etc.), so that the precision is known for values of recall equal to every hundredth unit between 0 and 1. mAP is the mean of average precision statistics computed independently for each class of objects.

We used the COCO API toolbox (Python) [23] to compute the above metrics and plot the graphs and curves. The experiments were carried out remotely on a shared Linux workstation with a 32-core Intel Xeon CPU and an NVIDIA GeForce GTX 1080 Titan GPU. We used PyTorch version 1.10 and `torchvision` version 0.11.1.

4.3 Models

We trained and evaluated several models of neural networks for object detection:

1. Faster R-CNN model (Sec. 2.3) from the `torchvision` library (PyTorch) [24]. Per [25] we took advantage of transfer learning – the provided backbone layers trained on ImageNet were frozen during our training.

2. YOLOv3 model (Sec. 2.4.1) from a GitHub repository[1] [26]. We used transfer learning, too – we did not train the DarkNet backbone module, which had been trained on ImageNet.

3. Our modification[2] of the previous YOLOv3 model that supports variable anchor box sizes (Sec. 3.2).

4. Three U-Net models from a GitHub repository [27] we modified for the heatmap regression task (Sec. 3.4). We parameterized the heatmaps by $\mathbf{\Sigma}$, the covariance matrix, which we set to $\mathbf{I}$, $10\mathbf{I}$, and $50\mathbf{I}$ ($\mathbf{I}$ is the 2×2 identity matrix). The models were trained from scratch.

5. An alternative (optimized) YOLOv3 model trained on a different (large) dataset of images similar to ours. More detailed information on its specification can be found in [28] (unpublished as of this writing). In particular, the images were taken in various environments with different camera setups.
   Apart from the usual way of having the model process the whole image at once, we were recommended to evaluate the model using a different evaluation strategy – divide the input images into smaller parts (we used 18) and have the same model process them simultaneously. In the report, we refer to these models either as "YOLOv3 eForce" or as "YOLOv3 eForce 18" when the multi-processing approach was used.

Unless otherwise stated, we trained the models ourselves on a training subset (80%) of the dataset by optimizing the default losses using the Adam optimizer

[1]We created a fork of the repository because we found training a custom model using the original code malfunctioning. It is available at `https://github.com/matejsuchanek/YOLOv3-in-PyTorch`.

[2]Code: `https://github.com/matejsuchanek/YOLOv3-in-PyTorch/tree/var_anchors` (`var_anchors` branch). See also Appendix A.3.

[29]. U-Net and Faster R-CNN were trained for 100 epochs, YOLOv3 for 200 epochs.

4.4 Evaluation

After training each model, we performed inference on the test split of our dataset (20%, ca. 140 images). We would always have the models evaluate the data with high sensitivity, that is, we decreased the threshold for considering an activation an actual detection to 0.01. We refer to the set of detections produced by each model as the "baseline".

Then, we applied our proposals to the baseline methods and produced a new set of detections for each application. In addition to a general comparison, we were particularly interested in the comparison of:

- YOLOv3 baseline to its modification that uses *variable anchor box sizes* (Sec. 3.2).
- Faster R-CNN, YOLOv3, YOLOv3 eForce, and YOLOv3 eForce 18 baseline detections to two derived sets of detections produced by:
  1. Strict application of location-based (horizon) and similarity-based (IoU) filters (see Sec. 3.3.1 and 3.3.2) that use knowledge about the scene geometry. Objects (bounding boxes) that would appear above the inferred horizon or that would not have at least 0.5 IoU[3] with the expected bounding box would be removed from the baseline set.
  2. Strict application of the location-based filter and "soft" application (Sec. 3.3.2) of the similarity-based filter. Detections that appear above the inferred horizon would be removed from the baseline set. No other detections would be removed, only their score would be updated by multiplying it by its IoU with the expected bounding box. (We refer to this as "combined score".)
- The three custom U-Net models to each other.

As a secondary metric, we also measured the time it takes the models to process their input. We performed the tests with multiple batch sizes to demonstrate how the performance changes in case we took advantage of multiprocessing.

4.5 Results

The results of our experiments are aggregated in Table 4.2 and visualized in Figures 4.1 to 4.11. We provide an analysis and a discussion of the results.

[3]The value of the threshold is further discussed in Sec. 4.5.4.

Class	Train data		Test data	
blue	2,533	(54.1%)	649	(53.0%)
yellow	1,896	(40.5%)	489	(40.0%)
red	251	(5.4%)	86	(7.0%)
All	4,680		1,224	

Table 4.1: Statistics over the number of samples of each class in the dataset. Counted separately for the train and test split.

4.5.1 General analysis

After breaking the results down by class, we noticed an oddity in the results (average precision) for the class of red cones. We realized this class was underrepresented in the dataset (ca. 6% of all samples, see Table 4.1 for complete data), and also, the samples were not consistent – some of the red cones would not stay on the ground and have the usual pose we assumed, but they were tumbled[4]. This discrepancy caused that although some cones were detected, the predictions either were not counted as true-positive or were not consistent with the geometrical information and thus filtered out (see further). An example of this situation is visualized in Fig. 4.14.

4.5.2 Performance

The comparison is shown in Fig. 4.1. YOLOv3 eForce was the fastest of all models, but with the worst accuracy. We could improve its accuracy using subimage multi-processing at the cost of making it ten times slower. U-Net model for heatmap regression was the slowest, but its accuracy was comparable to YOLOv3 when considering only the yellow and blue class. Though performance of YOLOv3 and Faster R-CNN may also seem comparable, we note the actual input size to YOLOv3 was 672×672 since YOLOv3 only accepts images of square shape.

4.5.3 Variable anchor box sizes for YOLOv3

We looked at the modified YOLOv3 architecture with variable anchor box sizes. This model suffered a significant drop in accuracy – the baseline model achieved mAP of 0.776, while the modified one's mAP was 0.614 (Fig. 4.4). We could not find a clear recurring trend in the yielded detections. Since both models were being trained on the same data with the same parameters, we believe variable anchor boxes make learning less stable and more difficult. Comparison of their output from a sample image is shown in Fig. 4.13.

[4]Ideally, those would be annotated as a different class that we would assign different parameters (i.e., the dimensions).

4.5.4 Direct application of scene geometry constraints

We also examined the results of the methods that use the knowledge about the scene geometry to filter the set of detections produced by a detector. We expected a noticeable increase in accuracy compared to baseline models (we now focused on AP for the yellow and blue cones), but it would mostly decrease when strict filters were used (see, e.g., R-CNN and YOLOv3 in Table 4.2), or decrease or just very slightly increase when the soft filter (combined scoring) was used. An exception was the YOLOv3 eForce 18 model (uniting detections from multiple portions of the input image), where combined scoring could increase AP for the blue class by more than 0.02 compared to the baseline algorithm (Fig. 4.6).

To understand the influence of the threshold on IoU of the strict filter, we relaxed it from 0.5 to 0.3 and re-evaluated the models. This time, the accuracy of most models recovered. Since increasing the threshold naturally decreases the number of positive detections and since it caused the accuracy (AP/mAP) to decrease, application of the filter decreased *recall* – the number of true-positive detections divided by the number of all detections in the ground truth (see also the definition in Sec. 4.2).

By an analysis of the dataset (Fig. 4.2), we discovered there was a high variance of the IoU of the annotations with the expected bounding boxes (according to the prior knowledge). The models were thus supposed to learn to predict bounding boxes that are not consistent with our model of prior knowledge. On the other hand, the strict filter relied on the prior knowledge, and therefore, its application was contra-productive and harmful because it removed valid detections. Alternatively, there might also have been imprecision in the camera model.

Furthermore, we revealed a cluster of outliers in the dataset. It was a subset of samples located less than 5 meters[5] from the camera with low IoU. We quickly identified the pattern – they represented cones located so close to the camera that only a part of them (the tip) was captured in the image. We find such samples very problematic because a) it is difficult to annotate them reliably, b) some predictors, like YOLOv3, may not be allowed to predict the object center outside the input image, and c) if annotated within the image bounds, they do not correspond to the prior knowledge model.

In summary, deep neural networks are known to be very powerful and capable of learning features of a very high level. In our case, they probably learned not to provide invalid detections, which naturally are not included in the training data. However, we observed an improvement in the model that was trained on data from various different environments. This makes us believe that the method of filtering detections could be more suitable in situations where there are little or no data from the target environment available to adapt a model to it. We also recommend that annotations of the training data be made with respect to the camera model.

[5]The distance was inferred using eq. (3.2) and the ray-casting algorithm (Sec. 3.2, page 13) based on the camera model.

4.5.5 U-Net models for heatmap regression

Finally, we were interested in the performance of our proposed U-Net models for heatmap regression. The model trained on heatmaps with the lowest variance ($\mathbf{\Sigma} = \mathbf{I}$) could not learn the representation and did not correctly predict any detections. The other two models ($\mathbf{\Sigma} \in \{10\mathbf{I}, 50\mathbf{I}\}$) did not outperform the other state-of-the-art architectures in precision and performance, but still achieved reasonable accuracy, with AP on the blue and yellow class around 0.8. The best of the three models trained on heatmaps with $\mathbf{\Sigma} = 50\mathbf{I}$ even succeeded in detecting some samples from the underrepresented class of red cones (Fig. 4.7). This model also predicted most of the identified objects (yellow and blue cones) with very high confidence (Fig. 4.9).

4.5.6 Plots

Figures 4.3 to 4.7 show plots of precision-recall curves in pairs. In these figures, higher recall and precision mean better accuracy. The upper plot always compares mean precision-recall dependence of the baseline method against methods enhanced by our proposals aggregating all three classes. The bottom plot compares the precision-recall dependence per method (discriminated by line style) and class (includes the “aggregate” class, discriminated by color). For example, in Fig. 4.5, bottom plot, the yellow dashed line intersects the point (0.7, 0.8) – if we set the threshold on the combined score so that the model could detect 70% of all objects, 80% of its produced detections would be valid.

Figures 4.8 and 4.9 display “scoring” curves in addition to the precision-recall curves. They show the value of the threshold (dotted line) as a function of recall. For example, in Fig. 4.8, middle plot, the blue dotted line intersects the point (0.6, 0.8) and the precision at the 60% recall is 85%. That is, if we set the objectness threshold to 0.8, the model would detect 60% of all objects, and 85% of its produced detections would be valid.

Model	mAP	Average precision blue	yellow	red	Time/batch (ms) batch size: 1/2/4
Faster R-CNN					32.2/53.0/97.4
baseline	**0.891**	**0.937**	**0.960**	**0.776**	
filtered (IoU $>$ 0.3)	0.876	0.937	0.960	0.731	
filtered (IoU $>$ 0.5)	0.823	0.928	0.944	0.597	
combined score	0.887	0.929	0.958	0.774	
YOLOv3					30.2/51.2/99.8
baseline	0.776	0.867	0.829	0.630	
filtered (IoU $>$ 0.3)	**0.777**	**0.869**	**0.830**	**0.631**	
filtered (IoU $>$ 0.5)	0.714	0.843	0.798	0.500	
combined score	0.773	0.867	0.822	0.630	
var. anchor boxes	0.614	0.707	0.705	0.431	
YOLOv3 eForce					5.0/6.5/11.7
baseline	0.377	0.425	0.532	0.174	
filtered (IoU $>$ 0.3)	0.379	0.426	0.533	0.179	
filtered (IoU $>$ 0.5)	0.373	0.417	0.510	**0.194**	
combined score	**0.384**	**0.430**	**0.535**	0.187	
YOLOv3 eForce 18					41.6/–/–
baseline	0.579	0.628	0.708	0.401	
filtered (IoU $>$ 0.3)	0.580	0.630	0.712	0.399	
filtered (IoU $>$ 0.5)	0.565	0.630	0.715	0.349	
combined score	**0.594**	**0.650**	**0.722**	**0.410**	
U-Net					44.1/75.9/155.6
$\mathbf{\Sigma} = \mathbf{I}$	0.0	0.0	0.0	0.0	
$\mathbf{\Sigma} = 10\mathbf{I}$	0.533	**0.814**	0.786	0.0	
$\mathbf{\Sigma} = 50\mathbf{I}$	**0.641**	0.796	**0.814**	**0.313**	

Table 4.2: Accuracy and performance of all tested models and methods on the test dataset. We show the mean average precision (mAP) as well as the average precision for each class. More granular results or other comparisons are presented in the following graphs.

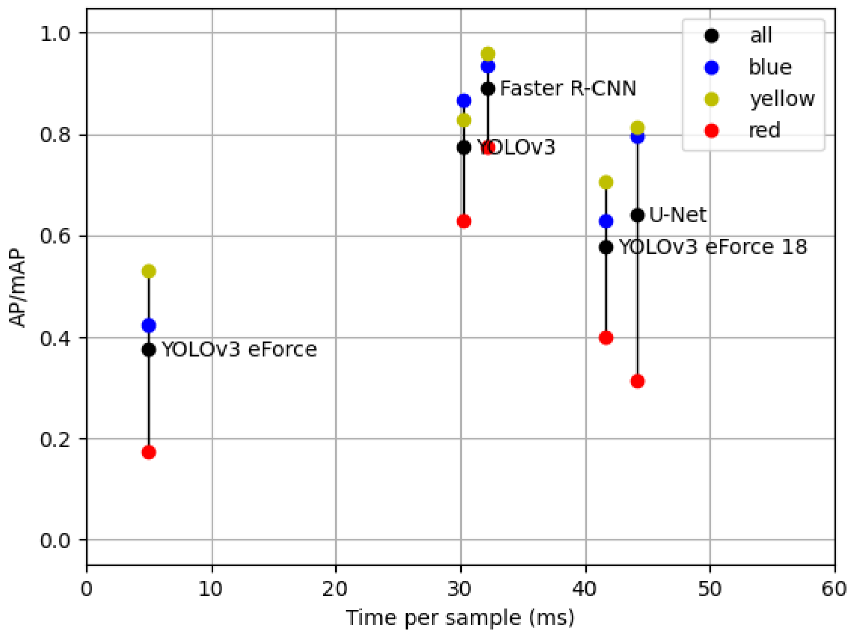


Figure 4.1: Visualization of measurement results from Table 4.2. We plot the performance (latency) versus accuracy (total and per class) of the baseline models. Faster architectures are on the left, more accurate ones are higher.

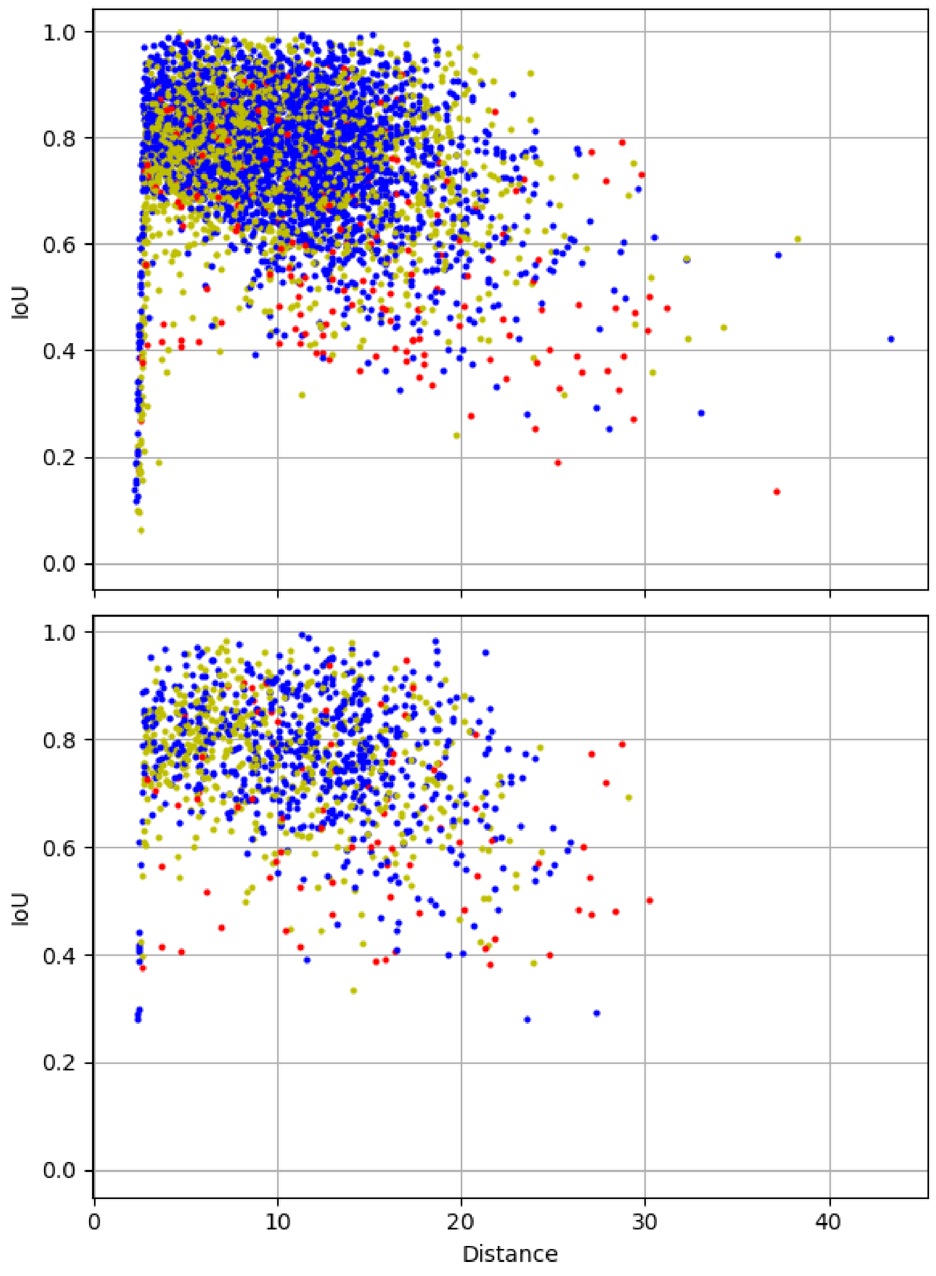
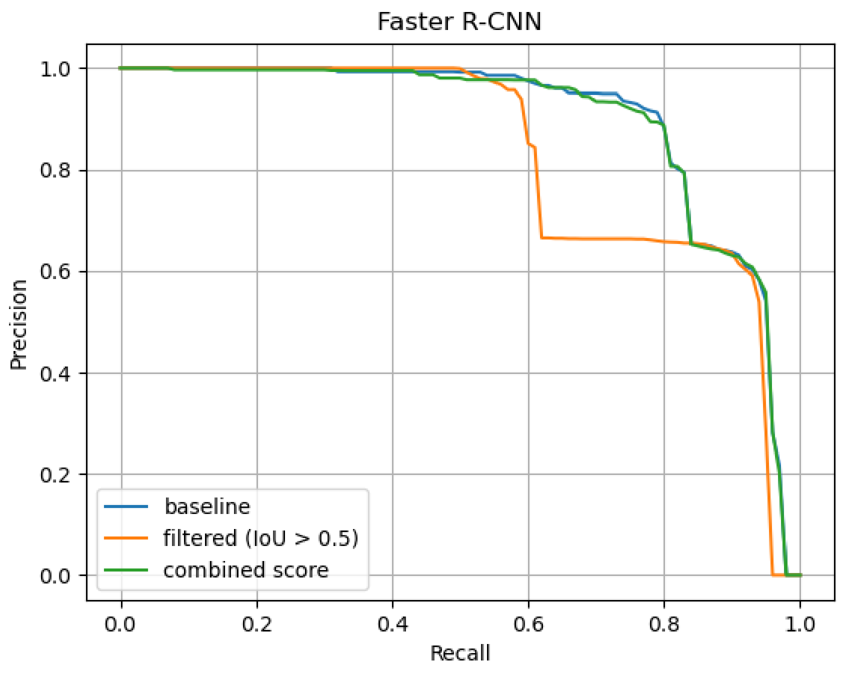


Figure 4.2: Scatter plot of all samples in the train (top) and test (bottom) dataset split. The x-axis indicates the inferred distance, y-axis their IoU with the expected bounding box according to the prior knowledge. We can see significant variance in IoU, cluster of outliers (objects very close to camera) and low IoU for red cones.



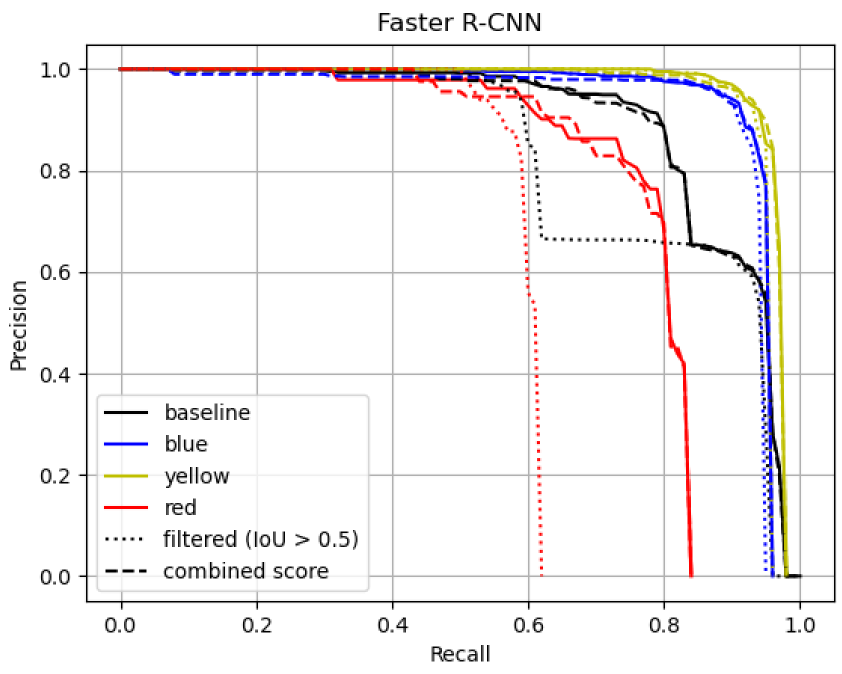
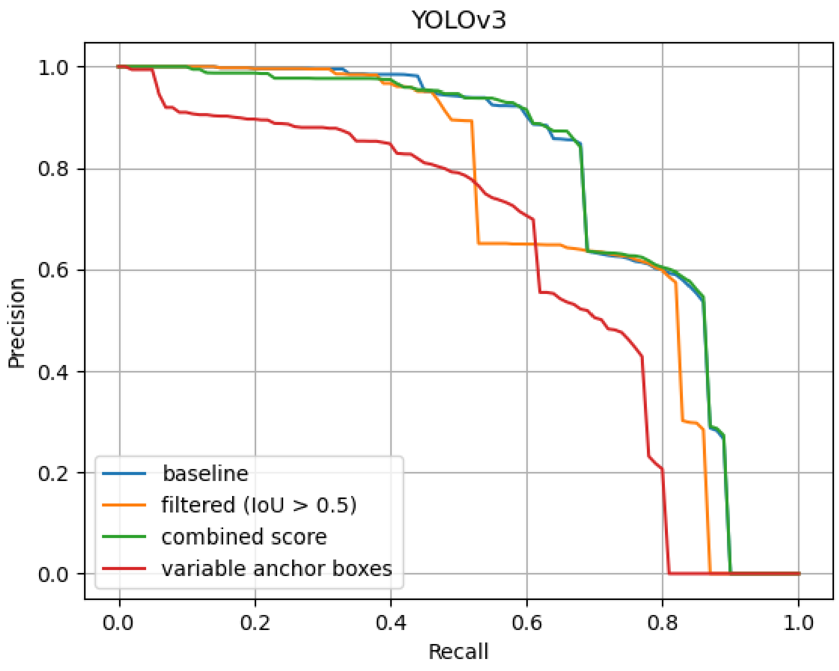


Figure 4.3: Faster R-CNN precision-recall curves (Sec. 4.5.6). Faster R-CNN had the best accuracy of all evaluated detectors, and our proposed methods for using prior knowledge did not improve it further.



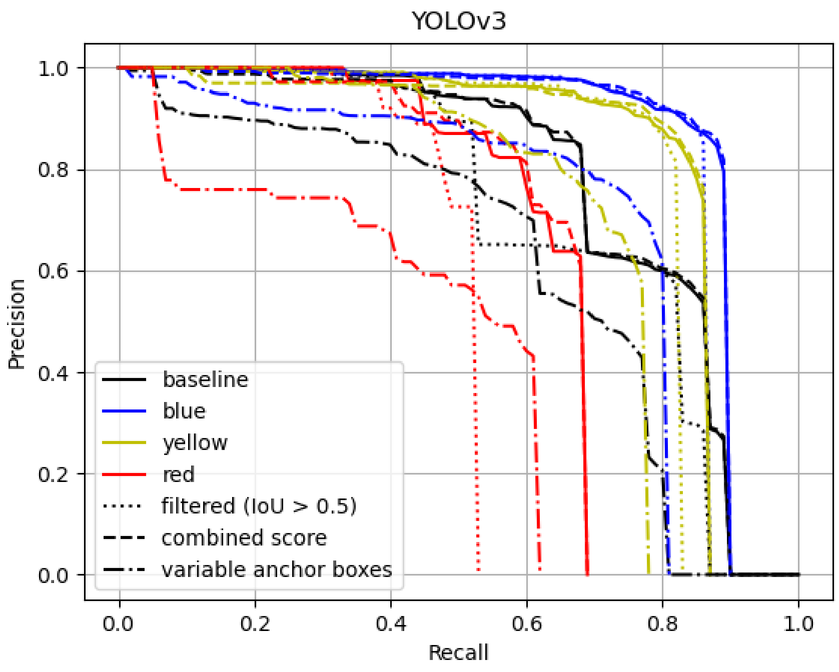
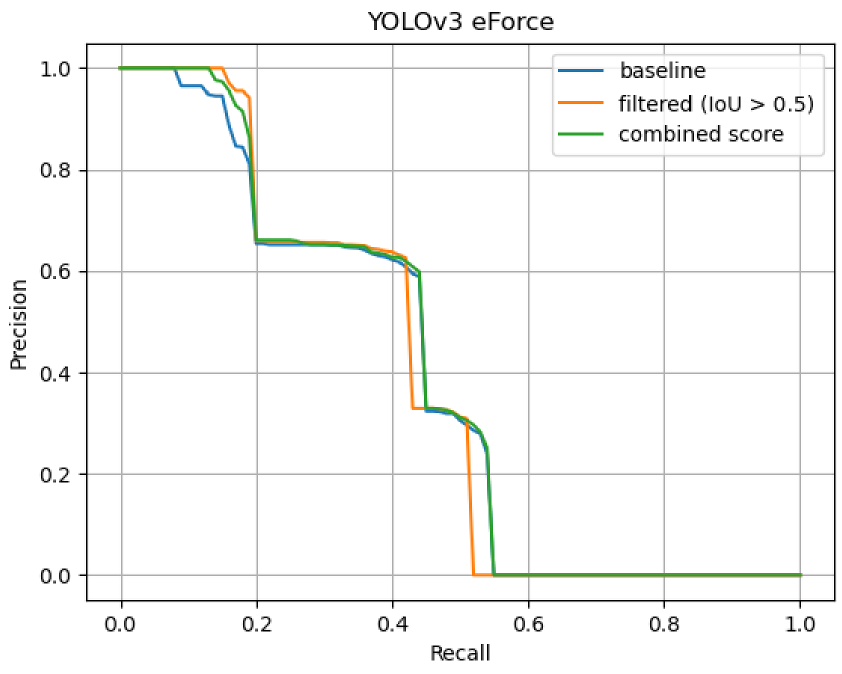


Figure 4.4: YOLOv3 precision-recall curves. We notice significant drop in accuracy of the modified version which uses variable anchor box sizes.



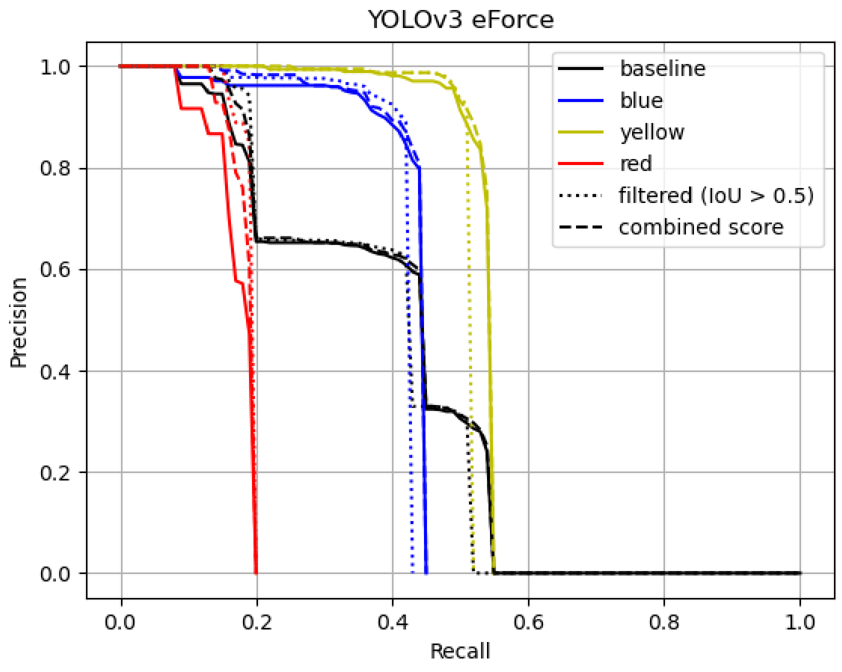
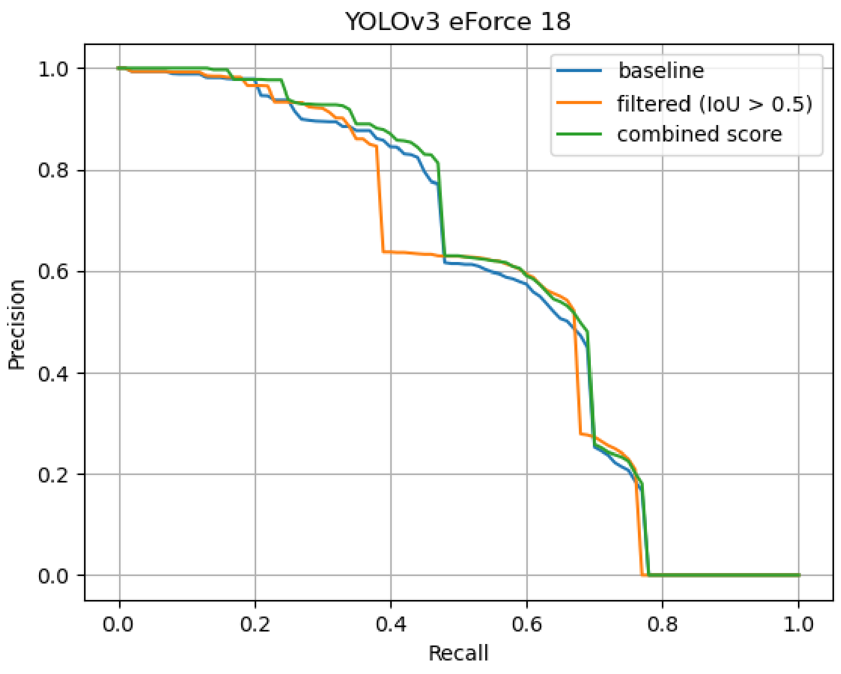


Figure 4.5: "YOLOv3 eForce" precision-recall curves. This detector had the worst accuracy with mAP = 0.38. Our methods could only improve it by less than 0.01.



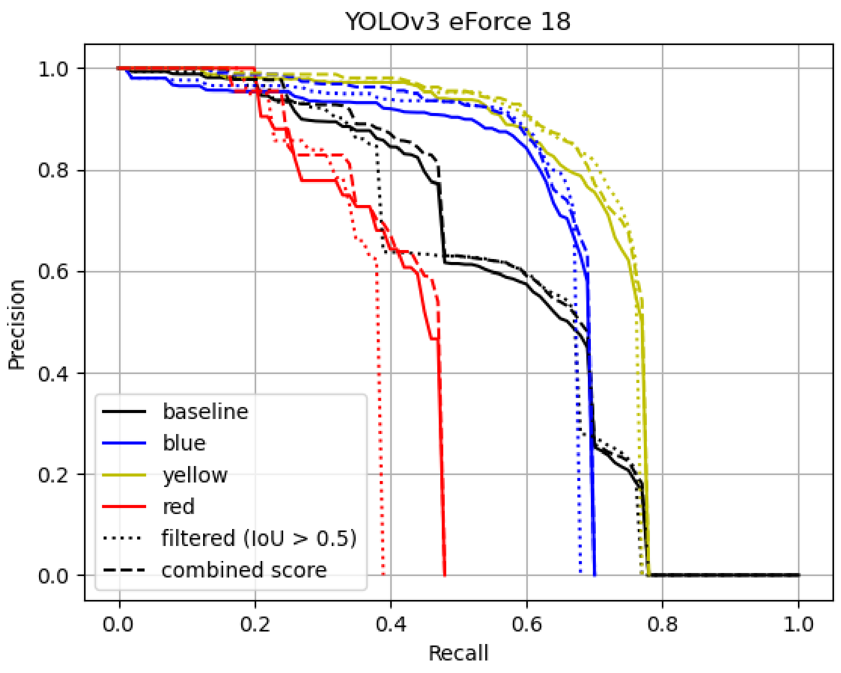
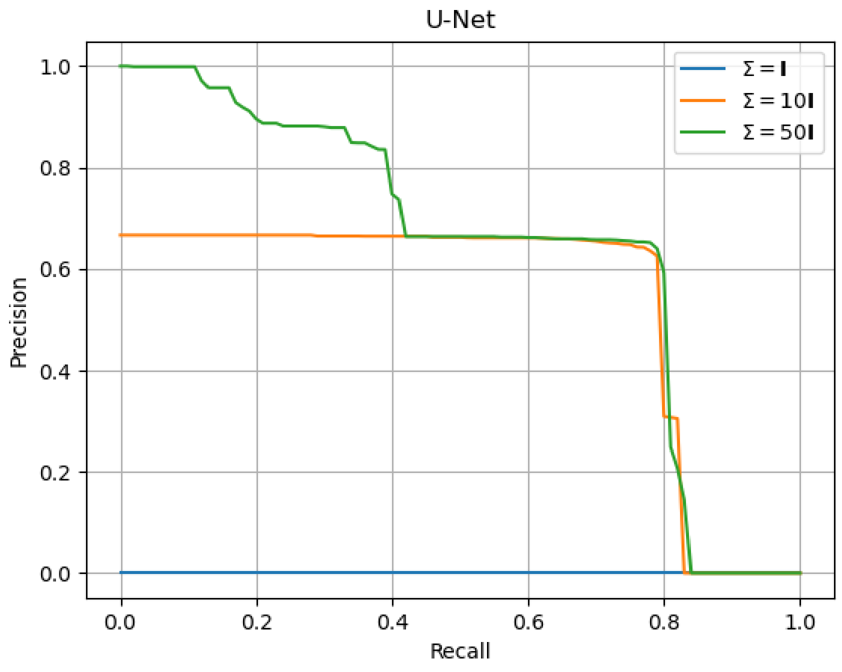


Figure 4.6: "YOLOv3 eForce 18" precision-recall curves. The filters could improve accuracy (mAP) of this model by ca. 0.015, though combined scoring on the blue class achieved even more than 0.02 increase in AP.



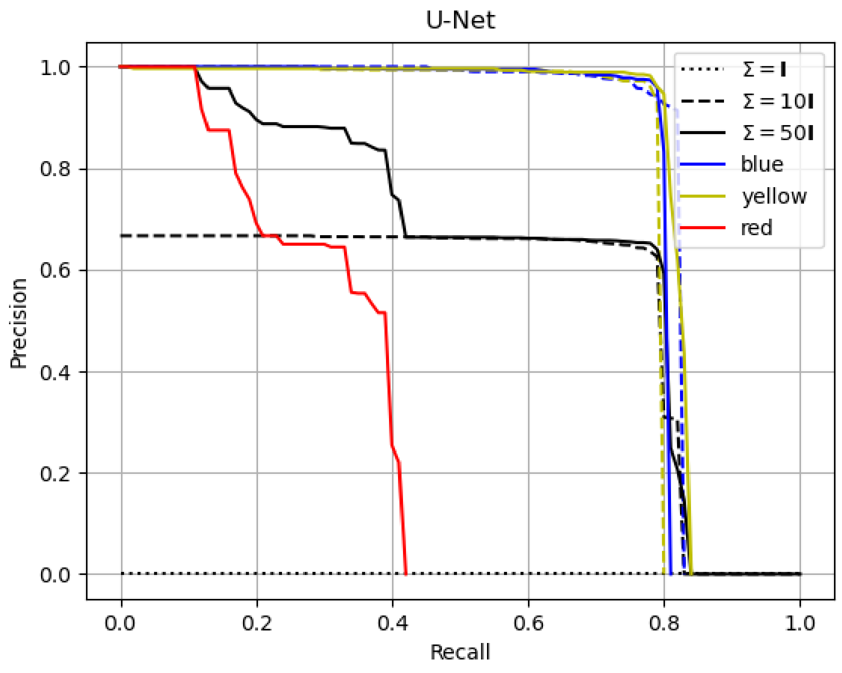


Figure 4.7: Precision-recall curves of U-Net for heatmap regression. The model using unit covariance did not learn to predict any objects and its accuracy (mAP) was zero. The accuracy of the other two models was comparable considering the yellow and blue class; one of the models could even detect some red cones (whereas the other model did not detect any).

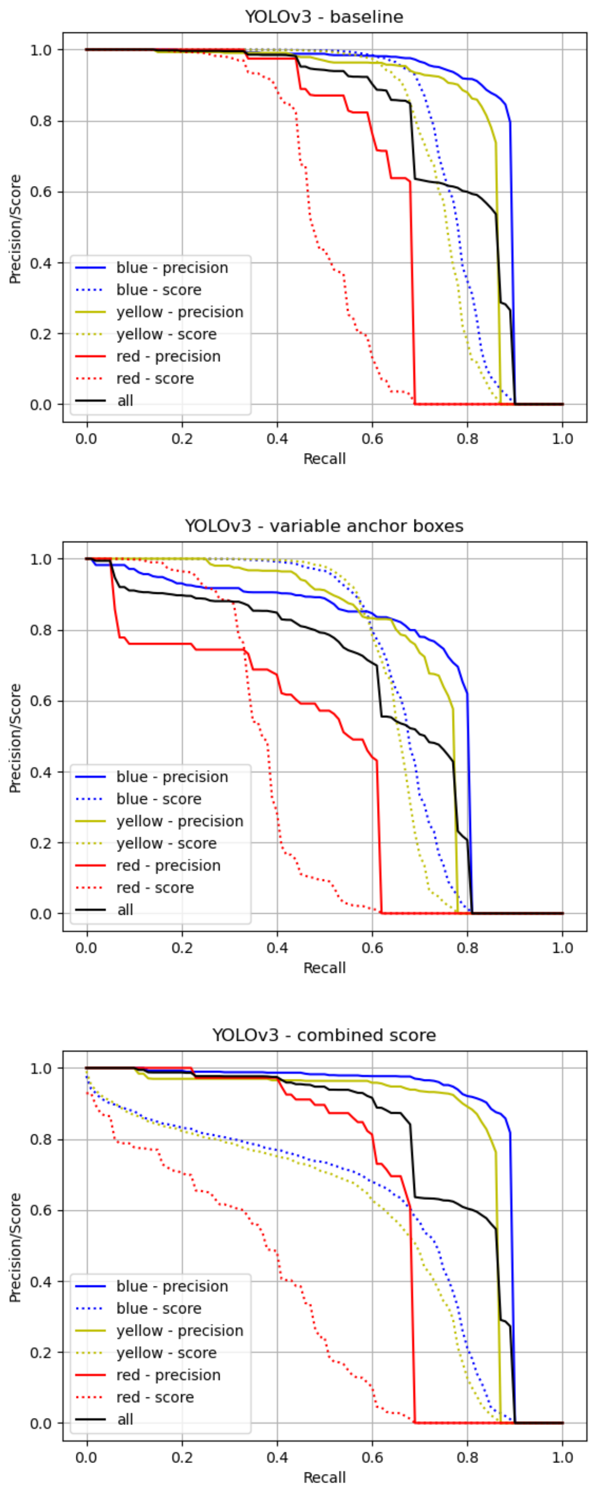
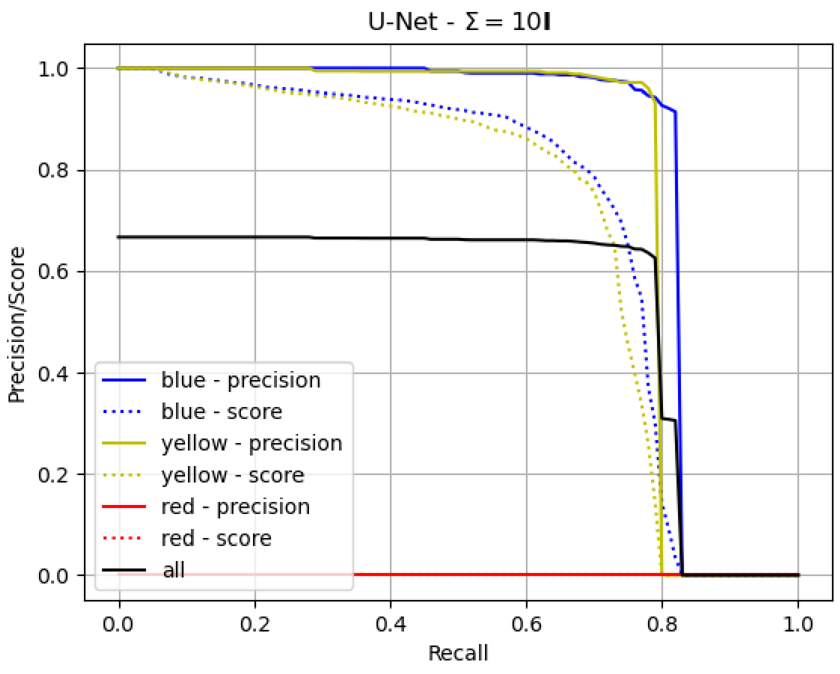


Figure 4.8: Scoring and precision-recall curves (see Sec. 4.5.6) of (top to bottom) baseline YOLOv3, modified YOLOv3 with variable anchor box sizes and baseline YOLOv3 with combined scoring.



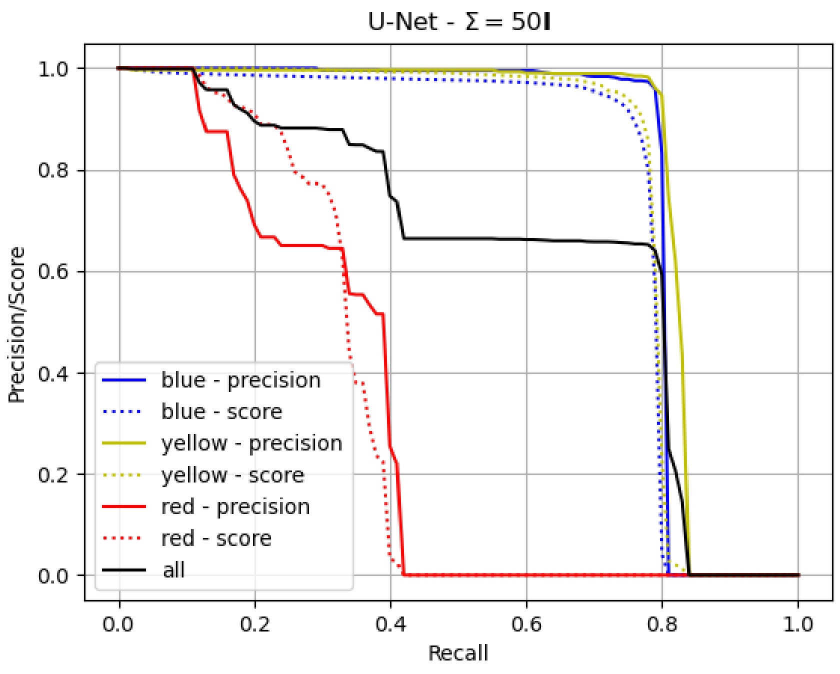


Figure 4.9: Scoring and precision-recall curves of the two best U-Net models. We notice the second model (bottom) yielded most detections with a very high confidence (blue and yellow dotted lines).

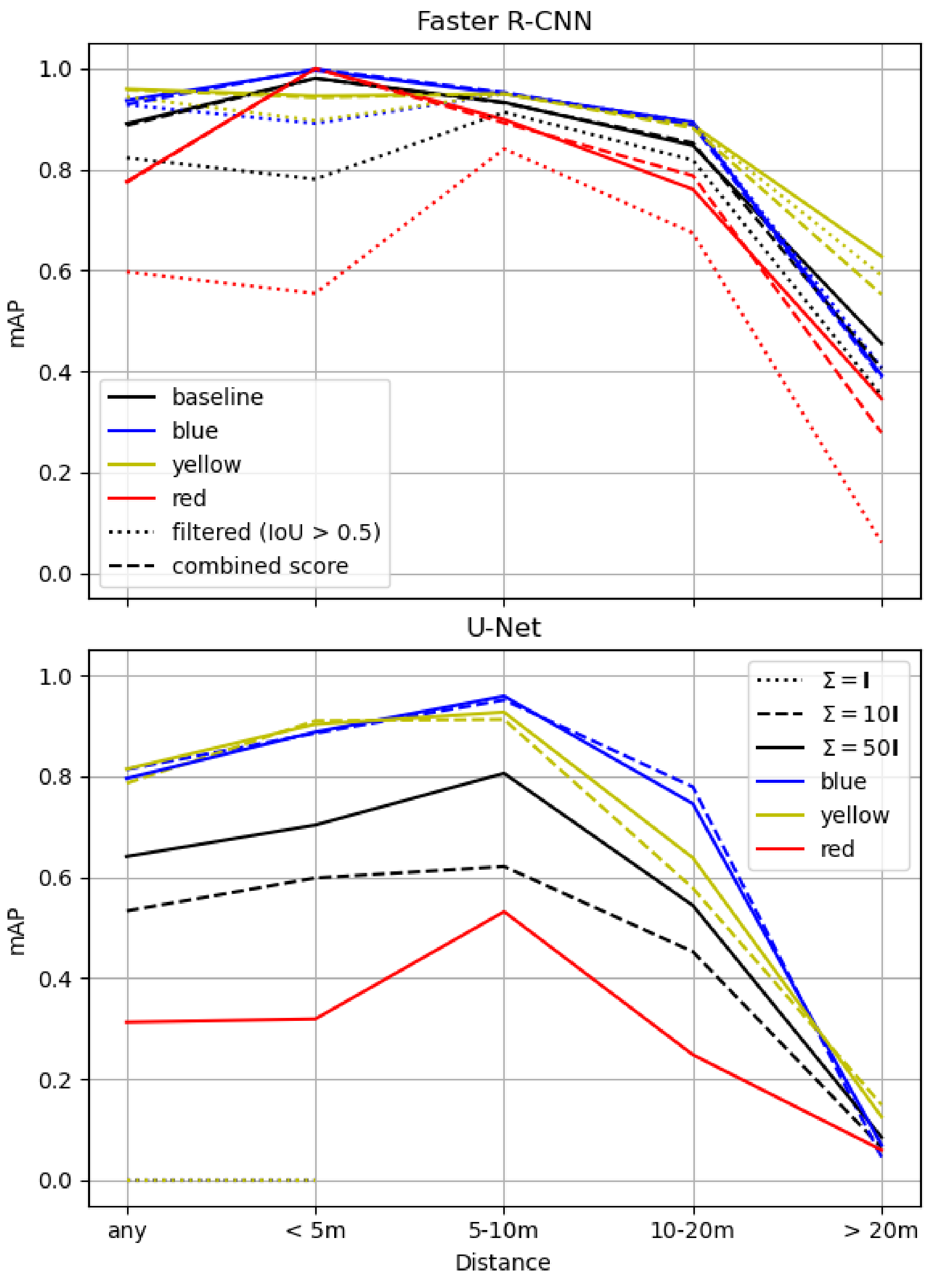


Figure 4.10: Accuracy of models grouped by distance of objects. The plots continue in Fig. 4.11.

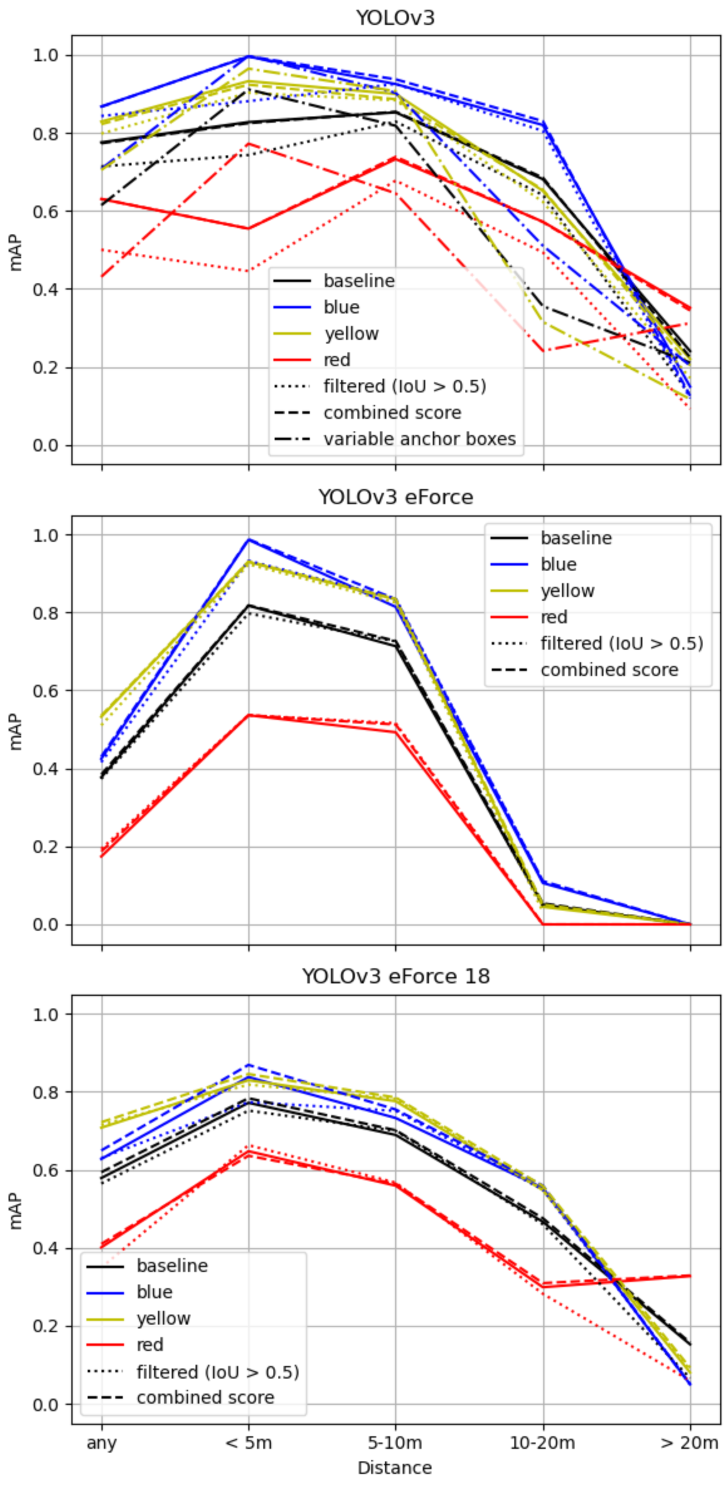


Figure 4.11: Continuation of Fig. 4.10. The two figures demonstrate that the detectors had problems with detecting more distant objects. However, there were only few objects more than 20 meters from the camera, both in training and test data (see Fig. 4.2).

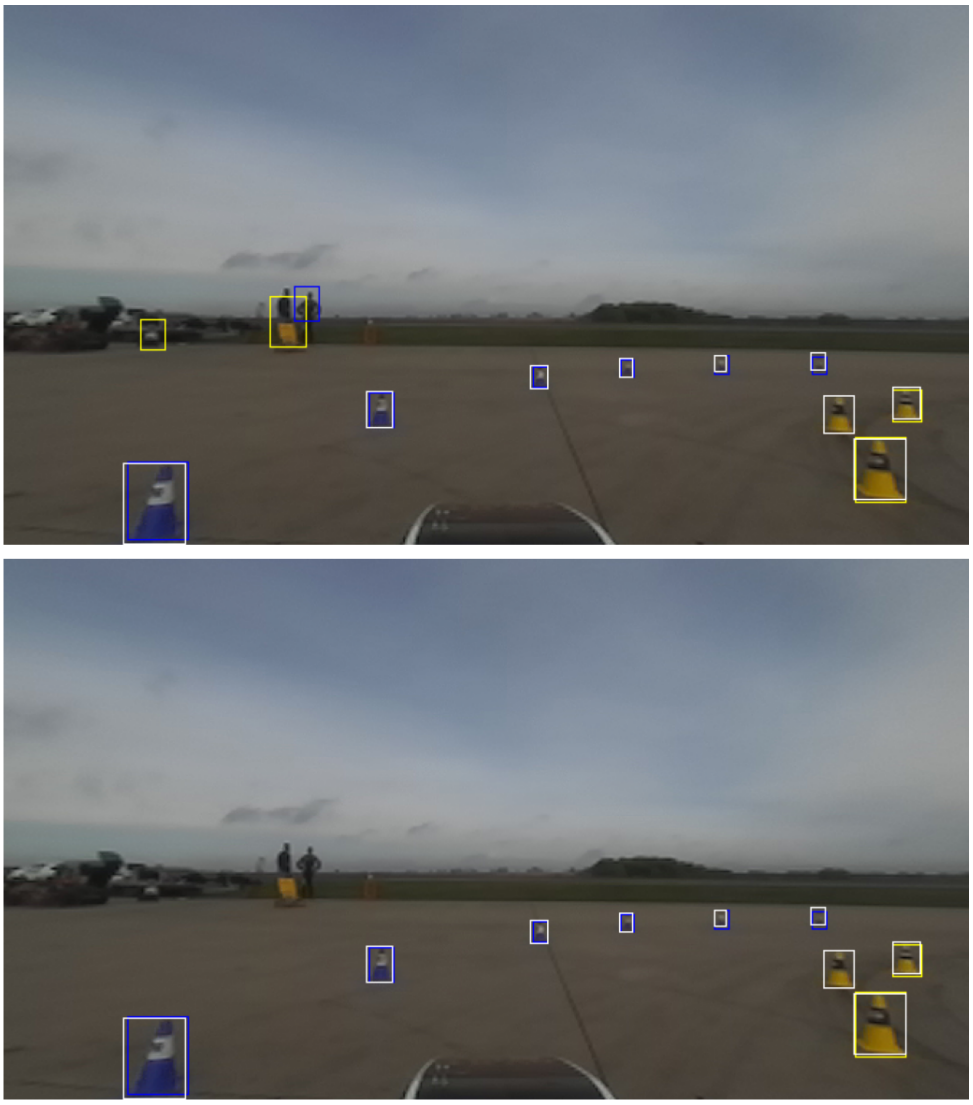


Figure 4.12: Demonstration of a successful application of the geometrical location- and similarity-based filters. The top image shows the predicted detections, the bottom one shows detections after the filters have been applied. White rectangles correspond to the ground truth, other bounding boxes correspond to the predictions and have the respective color (class).

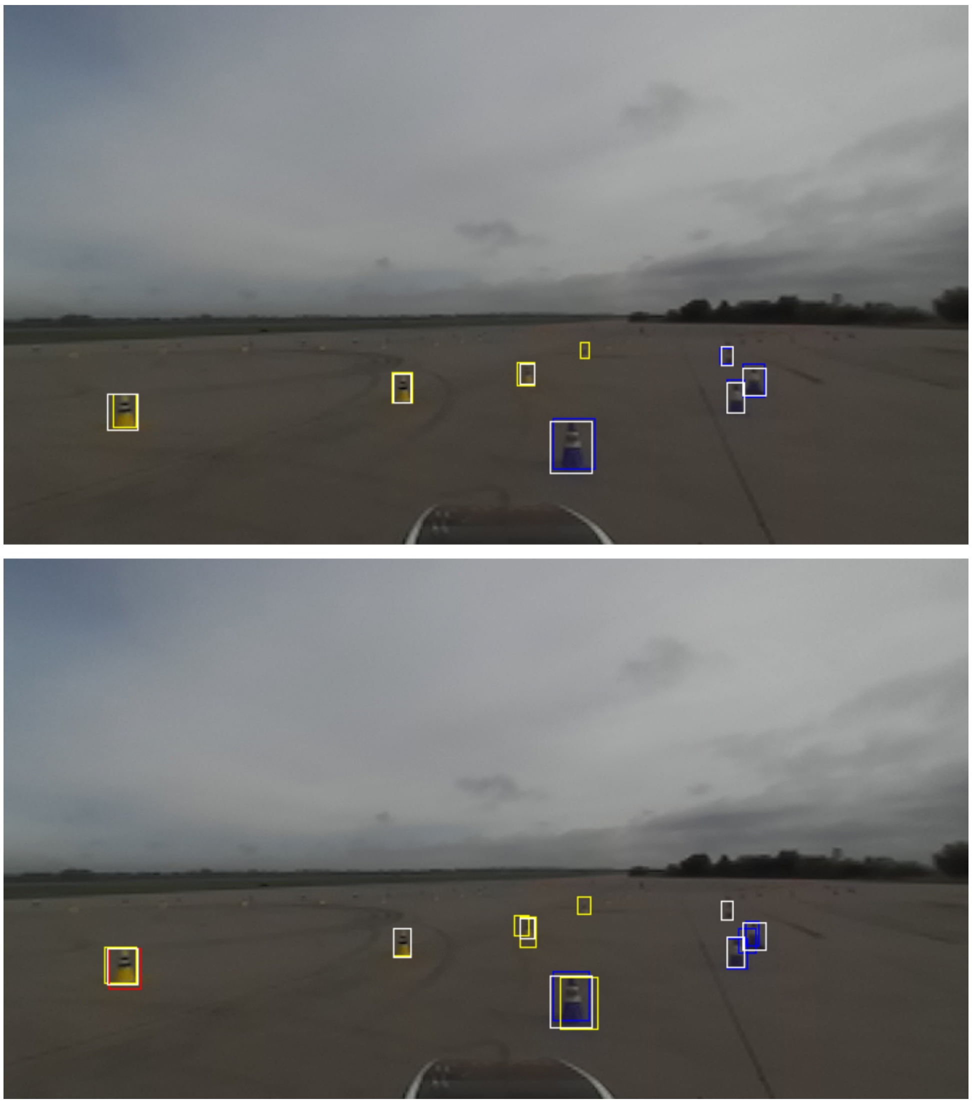


Figure 4.13: Sample image from the dataset evaluated by the YOLOv3 baseline model (top) and YOLOv3 with variable anchor box sizes (bottom). We can see rather suboptimal results in the bottom image. White rectangles correspond to the ground truth, other bounding boxes correspond to the predictions and have the respective color (class).

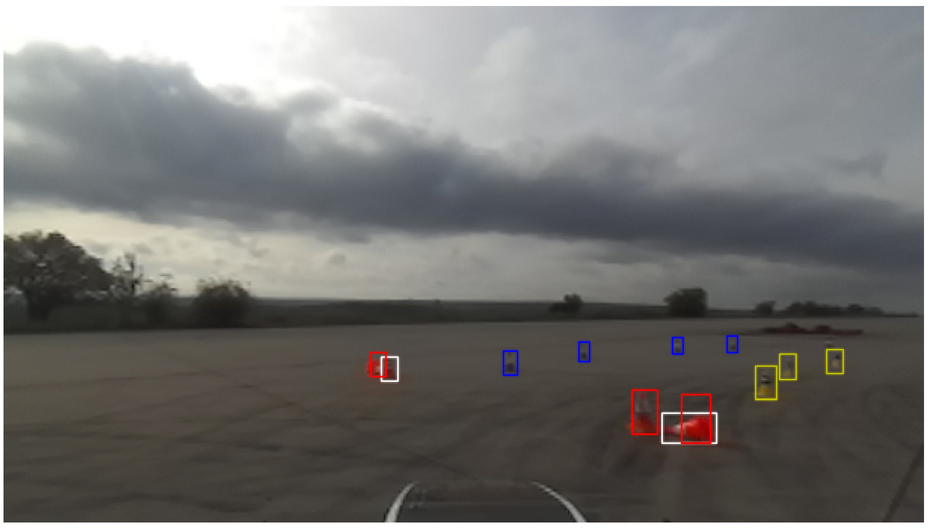


Figure 4.14: Demonstration of the problematic definition of red cones. Although the U-Net model could detect the one close to the bottom-right corner, the inferred bounding box is orthogonal to its annotation, and they overlap only partially. The detection thus does not count as true positive, but false positive (and the ground truth counts as false negative), which causes decrease in average precision.

Chapter 5

Conclusion

The goal of the thesis was to explore ways of incorporating additional information about the target environment into contemporary methods for object detection. We described the problem, reviewed existing algorithms, and presented their properties and design. We also searched for outcomes from related research fields and identified ideas that could help us in our efforts. Furthermore, we presented fundamental knowledge, concepts, and relationships from the field of computer vision and used them to derive techniques useful for our further work.

As the next step, we described our conception of the additional prior knowledge and proposed its formal representation. Then, we proposed three diverse improvements that take our findings into consideration. We suggested encoding the geometric information directly into a deep neural network or adding a new component to the detection pipeline that tries to identify and suppress impossible predictions. We also repurposed an existing neural network architecture and designed a different evaluation process that relies on prior information about the environment.

Finally, we created prototypes of the enhanced algorithms and compared their performance with the respective baseline methods that were not informed of the environment. We considered accuracy to be the key indicator. We found that the baseline methods are robust and their accuracy cannot be greatly improved through our proposals. However, we also revealed some isolated cases where the informed methods were more accurate.

We conclude that deep neural networks can probably learn their own representation of prior knowledge from the training data, which prevents them from making impossible decisions. We also express the hypothesis that our proposals might be more useful in situations where only few data about the target environment are available.

Bibliography

[1] Joseph Redmon and Ali Farhadi. YOLO9000: Better, faster, stronger. In *30TH IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR 2017)*, IEEE Conference on Computer Vision and Pattern Recognition, pages 6517–6525. IEEE; IEEE Comp Soc; CVF, 2017. 30th IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, JUL 21-26, 2017.

[2] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[3] Paul Viola and Michael Jones. Robust real-time object detection. In *International Journal of Computer Vision*, 2001.

[4] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR)*, IEEE Conference on Computer Vision and Pattern Recognition, pages 580–587. Comp Vis Fdn; IEEE; IEEE Comp Soc, 2014. 27th IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Columbus, OH, JUN 23-28, 2014.

[5] Rohith Gandhi. R-CNN, Fast R-CNN, Faster R-CNN, YOLO — object detection algorithms. `https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e`, 2018.

[6] Ross Girshick. Fast R-CNN. In *2015 IEEE INTERNATIONAL CONFERENCE ON COMPUTER VISION (ICCV)*, IEEE International Conference on Computer Vision, pages 1440–1448. Amazon; Microsoft;

Sansatime; Baidu; Intel; Facebook; Adobe; Panasonic; 360; Google; Omron; Blippar; iRobot; Hiscene; nVidia; Mvrec; Viscovery; AiCure, 2015. IEEE International Conference on Computer Vision, Santiago, CHILE, DEC 11-18, 2015.

[7] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In C Cortes, ND Lawrence, DD Lee, M Sugiyama, and R Garnett, editors, *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 28 (NIPS 2015)*, volume 28 of *Advances in Neural Information Processing Systems*, 2015. 29th Annual Conference on Neural Information Processing Systems (NIPS), Montreal, CANADA, DEC 07-12, 2015.

[8] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR)*, IEEE Conference on Computer Vision and Pattern Recognition, pages 779–788. IEEE Comp Soc; Comp Vis Fdn, 2016. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, JUN 27-30, 2016.

[9] Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement, 2018. arXiv preprint arXiv:1804.02767.

[10] Adrian Bulat and Georgios Tzimiropoulos. How far are we from solving the 2d & 3d face alignment problem? (and a dataset of 230,000 3d facial landmarks). In *2017 IEEE INTERNATIONAL CONFERENCE ON COMPUTER VISION (ICCV)*, IEEE International Conference on Computer Vision, pages 1021–1030. IEEE; IEEE Comp Soc, 2017. 16th IEEE International Conference on Computer Vision (ICCV), Venice, ITALY, OCT 22-29, 2017.

[11] Marek Kowalski, Jacek Naruniec, and Tomasz Trzcinski. Deep alignment network: A convolutional neural network for robust face alignment. In *2017 IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION WORKSHOPS (CVPRW)*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, pages 2034–2043. IEEE; IEEE Comp Soc; CVF, 2017. 30th IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Honolulu, HI, JUL 21-26, 2017.

[12] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In B Leibe, J Matas, N Sebe, and M Welling, editors, *COMPUTER VISION - ECCV 2016, PT VIII*, volume 9912 of *Lecture Notes in Computer Science*, pages 483–499, 2016. 14th European Conference on Computer Vision (ECCV), Amsterdam, NETHERLANDS, OCT 08-16, 2016.

[13] Jonathan Tompson, Arjun Jain, Yann LeCun, and Christoph Bregler. Joint training of a convolutional network and a graphical model for human

pose estimation. In Z Ghahramani, M Welling, C Cortes, ND Lawrence, and KQ Weinberger, editors, *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 27 (NIPS 2014)*, volume 27 of *Advances in Neural Information Processing Systems*, 2014. 28th Conference on Neural Information Processing Systems (NIPS), Montreal, CANADA, DEC 08-13, 2014.

[14] Zhusi Zhong, Jie Li, Zhenxi Zhang, Zhicheng Jiao, and Xinbo Gao. An attention-guided deep regression model for landmark detection in cephalograms. In D Shen, T Liu, TM Peters, LH Staib, C Essert, S Zhou, PT Yap, and A Khan, editors, *MEDICAL IMAGE COMPUTING AND COMPUTER ASSISTED INTERVENTION - MICCAI 2019, PT VI*, volume 11769 of *Lecture Notes in Computer Science*, pages 540–548, 2019. 10th International Workshop on Machine Learning in Medical Imaging (MLMI) / 22nd International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI), Shenzhen, PEOPLES R CHINA, OCT 13-17, 2019.

[15] Hei Law and Jia Deng. CornerNet: Detecting objects as paired keypoints. In V Ferrari, M Hebert, C Sminchisescu, and Y Weiss, editors, *COMPUTER VISION - ECCV 2018, PT XIV*, volume 11218 of *Lecture Notes in Computer Science*, pages 765–781, 2018. 15th European Conference on Computer Vision (ECCV), Munich, GERMANY, SEP 08-14, 2018.

[16] Xingyi Zhou, Jiacheng Zhuo, and Philipp Krahenbuhl. Bottom-up object detection by grouping extreme and center points. In *2019 IEEE/CVF CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR 2019)*, IEEE Conference on Computer Vision and Pattern Recognition, pages 850–859. IEEE; CVF; IEEE Comp Soc, 2019. 32nd IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, JUN 16-20, 2019.

[17] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Lecture Notes in Computer Science*, pages 234–241. Springer International Publishing, 2015.

[18] Wikipedia contributors. Pinhole camera model — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Pinhole_camera_model&oldid=1028923582`, 2021. [Online; accessed 12-January-2022].

[19] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2 edition, 2004.

[20] Wikipedia contributors. Camera resectioning — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Camera_resectioning&oldid=1062633321`, 2021. [Online; accessed 12-January-2022].

[21] Wikipedia contributors. Homogeneous coordinates — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Homogeneous_coordinates&oldid=1062485619`, 2021. [Online; accessed 12-January-2022].

[22] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *Computer Vision – ECCV 2014*, pages 740–755. Springer International Publishing, 2014. [Online: `http://cocodataset.org/#download`].

[23] Piotr Dollar and Tsung-Yi Lin. COCO API. `https://github.com/cocodataset/cocoapi/`, 2020.

[24] torchvision.models. PyTorch documentation. `https://pytorch.org/vision/0.11/models.html` [Online; accessed on 2022-05-14].

[25] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *2014 IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR)*, IEEE Conference on Computer Vision and Pattern Recognition, pages 1717–1724. Comp Vis Fdn; IEEE; IEEE Comp Soc, 2014. 27th IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Columbus, OH, JUN 23-28, 2014.

[26] Haoyu Wu. YOLOv3 in PyTorch. `https://github.com/westerndigitalcorporation/YOLOv3-in-PyTorch`, 2019.

[27] U-Net: Semantic segmentation with PyTorch. `https://github.com/milesial/Pytorch-UNet`, 2017.

[28] Roman Šíp. Visual detection of traffic cones for autonomous student formula, 2022. [Thesis to be defended].

[29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. arXiv preprint arXiv:1412.6980.

[30] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.

Appendix A
Contents of CD

A.1 List of files

- `dataset1.zip`
- `dataset2.zip`
- `generator.py`
- `generate_anchors.py`

A.2 Custom synthetic dataset

In the early stage of our work, we created a small dataset (100 images, 600×1024), which imitates the environment we dealt with – path delimited by traffic cones on the road. We used Blender software [30] to create the scene and its Python API to simulate movement and obtain the ground truth. The images, ground truth annotations, and the source file are included in the `dataset1.zip` and `dataset2.zip` files.

A.3 Support for variable anchor box dimensions YOLOv3 model

The modified version of YOLOv3 that uses anchor boxes with variable sizes respecting the current camera model was published at `https://github.com/matejsuchanek/YOLOv3-in-PyTorch/tree/var_anchors`. It is stored in the `var_anchors` branch of our fork of the third-party YOLOv3 implementation in PyTorch that we found to be malfunctioning.

Sizes of the anchor boxes should be saved in the `anchor.py` file in the `src/` directory. They can be generated using the `generate_anchors.py` script and the `generator.py` class from the thesis attachment, which implement the ray-casting algorithm (Sec. 3.2, page 13) in steps. When run, the script produces the `anchors.py` file based on the `config` (camera model, see Sec. 3.1.1) and `anchors_example.png` image, similar to that in Fig. 3.2 (for visual feedback).