Detection of Traffic Scene Objects using YOLO Algorithm

Pascal Alain Dkengne Sielenou, Stéphane Girard

Detection of Traffic Scene Objects using YOLO Algorithm

Theory and Practical Guide

Pascal Alain Dkengne Sielenou & Stéphane Girard

Published by Inria de l'Université de Grenoble Alpes

First printing, January 2023

Contents

Introduction

Roads are dangerous since every year 1.35 million people die from traffic accidents and 94% of serious crashes are due to human errors (https://ultralytics.com/solutions/ai-in-self-driving-vehicles). Fortunately, advances in computer vision and artificial intelligence are bringing us closer to an exciting new era of transportation with autonomous vehicles that will reduce traffic congestion and increase road safety. Artificial intelligence has been introduced in modern vehicles for some time now. It all started with intelligent driving assistants that monitor the vehicle's surroundings and assist the drivers or alert them in case of emergency or risk of accident. To qualify as fully autonomous, a vehicle must be able to navigate without human intervention to a predetermined destination over roads that have not been adapted for its use. Autonomous vehicles implement automatic decision-making systems using artificial intelligence fed by streams of data from divers sensors such as cameras, LiDAR (Light Detection And Ranging), RADAR (Radio Detection And Ranging), GPS (Global Positioning System), or inertial sensors. The functioning of autonomous vehicles is based on the four key components, namely perception, localization, prediction and decision-making (https://neptune.ai/blog/self-driving-cars-with-convolutional-neural-networks-cnn). The decision-making includes path planning and vehicle motion control. Prediction consists in inferring the next actions of other road users. Perception refers to the ability of localizing and classifying all objects present in a vehicle's environment. Localization represents algorithms for calculating during driving the sequences of 3D coordinates for the positions and angles for the orientations of a vehicle with respect to the road principal axis.

The main goal of this work is to build a real-time traffic scene object detector for automated driving systems and their reliability analysis. The rest of this document is organized as follows. Chapter 1 recalls the basic concepts in object detection. Chapter 2 describes in detail the internal working of YOLO (You Only Look Once) algorithm for real-time object detection. In Chapter 3, we present the considered

classes of traffic scene objects along with the image labeling process. In Chapter 4, we illustrate the training stages of our object detector based on a model from the YOLO family.

1. Basic concepts in object detection

1.1 Introduction

Object detection is a computer vision technique aiming to generate predictions of both locations and classes of all target objects present in an image. Specifically, an object detector draws labeled bounding boxes around the detected objects, each label indicating the class of the object located within the corresponding bounding box. This is illustrated in Figure A.1, where an object detector identifies and locates all objects from the **cat** class and the **person** class. Note that when there are objects of different classes present in the scene described by a given image, an object detector can only identify all objects from the classes it was trained on. The objects from the unconsidered classes will be classified as background and no bounding box will be assigned to them.

Advanced object detection techniques are increasingly adopted and implemented in many practical technologies from various domains across the world to improve their effectiveness. The areas where these technologies can be found in action include autonomous vehicles, medical image analysis, video surveillance and image retrieval as one can see in [5, 14, 15, 23, 29]. The aforementioned use cases can be described in detail as follows. In autonomous vehicles, object detectors are used to localize various objects present in the traffic scene. This understanding of the surrounding environment of a vehicle can help the automated driving system to make decision for safe navigation. In medical image analysis, object detectors are used to help radiologists and other medical specialists to spent less time in the localization of lesions present in medical scans. This faster disease detection can considerably reduce the starting time of the treatment. In video surveillance, object detectors are used to localize intruders. This information about intrusions can be used to trigger an alarm or to alert the appropriate authorities. In image retrieval, object detectors identify objects in the query image and return from a large database of digital images those which are *similar* (for example, in terms of contents) to the query image.

An object detector have to solve the problem of classification and the problem of localization. The classification problem consists to identify objects of interest and to assign them the appropriate object class labels. The localization problem consists to assign accurate bounding boxes to all identified objects. Recent object detectors can be split into two major groups, namely the two-stage object detectors and the single-stage object detectors. The two-stage detectors use two independent processes which are optimized separately. The first process solves the localization problem approximately, that is, identifies potential bounding boxes that likely contain objects. The second process solves the classification problem and makes corrections to bounding boxes where objects of interest are identified. Such object detectors include the R-CNN [11], https://github.com/rbgirshick/rcnn, the Fast R-CNN [10], https://github.com/rbgirshick/fast-rcnn, the Faster R-CNN [27], https://github.com/rbgirshick/py-faster-rcnn, the Cascade R-CNN [4], https://github.com/zhaoweicai/cascade-rcnn and the R-FCN [6], https://github.com/daijifeng001/r-fcn. The single stage detectors use only one process to solve the classification problem and the localization problem in an unified way. Such detectors include the SSD [21], https://github.com/rykov8/ssd_keras, the FSSD, [17], https://github.com/lzx1413/PytorchSSD, RFB [20], https://github.com/GOATmessi7/RFBNet. the RetinaNet [19], https://github.com/fizyr/keras-retinanet and the YOLO family [3, 24, 25, 26, 31], https://github.com/ultralytics/yolov5, https://github.com/wongkinyiu/yolov7.

Although two-stage detectors are considered more accurate than single-stage object detectors, they have a slower inference speed than single-stage detectors. In this work, we focus on the YOLO family which is the most popular real-time object detector. Furthermore, this family has an active community of researchers and developers which constantly release new versions outperforming the previous ones in terms of both accuracy and detection time. Before describing the main architecture of the YOLO family, we introduce in the next section some important basic components and concepts common to all modern object detectors.

1.2 Bounding box

In object detection, a bounding box is usually used to describe the spatial location of an object from a class of interest within an image. A bounding box associated with a two-dimensional object can be defined as the minimum rectangle which contains that object. By convention, the (x,y)-coordinate system of an image has the origin $(0,0)$ at the upper left corner pixel of the entire image. In this coordinate system, a bounding box is usually described by the upper left corner pixel coordinates $(x_{\min}, y_{\min})$ and the lower right corner pixel $(x_{\max}, y_{\max})$. This is illustrated in Figure A.2 where coordinates of the bounding box associated with a **Coke in can** are indicated. Other commonly used representations for the bounding box coordinates in the (x,y)-coordinate system of an image consider as first components either the upper left corner pixel coordinates $(x_{\min}, y_{\min})$ or the bounding box center, and as last components the width and height of the bounding box. This is illustrated in Figure A.3 where bounding box coordinates associated with a **cat** is represented. We end this section by pointing out that in object detection modeling, annotating a training data set of images consists to

indicate the bounding box coordinates as well as the object class labels of all target objects which are present in the considered data set.

1.3 Intersection over Union (IoU)

Consider a target object to be detected represented by a ground-truth bounding box $\mathbf{B}_g$ (hand labeled bounding box from an annotated image) and the associated predicted bounding box $\mathbf{B}_p$ (generated bounding box by a detection model). Figure A.4 shows an example of such bounding boxes when the target object in an image belongs to the **stop sign** class. The Intersection over Union (IoU) between the two aforementioned bounding boxes is equal to the area of their overlap divided by the area of their union, that is

$$\text{IoU}\left(\mathbf{B}_g, \mathbf{B}_p\right) = \frac{\text{area}\left(\mathbf{B}_g \cap \mathbf{B}_p\right)}{\text{area}\left(\mathbf{B}_g \cup \mathbf{B}_p\right)}.$$

A visual illustration of the IoU is provided in Figure A.5. The IoU is equal to 1 when a perfect match occurs whereas the IoU is equal to 0 when the two involved bounding boxes do not intersect each other. The higher the value of the IoU, the better the detection is considered. It is worth noticing that as object detection also performs the classification of each bounding box, only ground-truth and predicted bounding boxes associated with the same class of objects are comparable through the IoU. By setting an IoU threshold, one can define the quality of a detector. For example, large thresholds generate detection of high quality as the predicted bounding boxes are required to be tightly align with the ground-truth bounding boxes. On the other hand, small thresholds generate detection of poor quality as the predicted bounding boxes may have small overlap with the ground-truth bounding boxes. Generally, the value of this threshold is kept at 0.5. But it is advised to iterate with different values to analyze the difference. Detection of increasing quality is shown in Figure A.6. One can see that there is very less overlap between bounding boxes as the value of the IoU increases. In the sequel, we will see that IoU is an important concept used in many tasks related to the modeling and the evaluation of an object detector.

1.4 Anchor box

Object detection algorithms usually define and explore a large number of bounding boxes having predefined dimensions in the input image. Then, select the bounding boxes which contains objects of interest. Finally, correct their coordinates to predict more accurate bounding boxes. Each bounding box defined during the exploration stage is called an **anchor box**. The difference between the selected anchor box and the corresponding ground-truth bounding box is called an **offset**. Illustrations of anchor boxes and offsets are shown in Figure A.7 in which the input image is divided into 2×1 bounding boxes, and in Figure A.8 in which the input image is divided into 3×2 bounding boxes.

Anchor boxes of various sizes or dimensions (widths and heights) are used during the exploration stage with the aim to find bounding boxes that are nearest to ground-truth bounding boxes. One of the strategies used to obtain candidate anchor boxes is based on the fact that most objects from the same class have a similar shape. For

example, a bounding box corresponding to an image of a person has a greater height than width, and a bounding box corresponding to an image of a car has a greater width than height. Consequently, by inspecting the ground-truth bounding boxes corresponding to objects from various classes, it is possible to identify the dimensions of the majority of bounding boxes present in the training data set. These dimensions can be obtained by employing k-means clustering on top of the training ground-truth bounding boxes. To reduce the model complexity, it is recommended to take $k = 5$ as the number of clusters. Anchor boxes to consider are the k resulting cluster centroids. It is important to note that this clustering uses the metric d defined by

$$\mathrm{d}\left(\mathrm{box}, \mathrm{centroid}\right) = 1 - \mathrm{IoU}\left(\mathrm{box}, \mathrm{centroid}\right).$$

Figure A.9 shows two anchor boxes where one is suitable for an object from a **person** class and the other is suitable for an object from the **car** class. One can see how close they are to the corresponding ground-truth bounding boxes. The important thing to know is that anchor boxes help object detectors to predict appropriate locations of objects at different scales by focusing on the predictions of offsets corresponding to anchor boxes to match with the ground-truth bounding boxes.

1.5 Non-Maximum Suppression (NMS)

Many objects from the same classes can take various shapes and can appear at different scales in images. To efficiently capture each of such objects, object detectors making use of anchor boxes generally associate to each object many overlapping bounding boxes. To remove redundant and less accurate predictions of locations, an algorithm called Non-Maximum Suppression is used. **Non-Maximum Suppression** (NMS) is a technique commonly used in object detection to select from overlapping bounding boxes the most appropriate bounding box to associated with each object. The NMS algorithm assumes that bounding boxes and the corresponding object classes along with the confidence scores are known. The confidence score of a bounding box is a numerical value which quantifies how certain the model is that the desired object is present in that bounding box. The confidence score is also referred to as the prediction probability. The NMS technique is described in Algorithm 1. For illustration, we consider the image provided in Figure A.10 in which redundant and overlapping bounding boxes together with their confidence scores are displayed for an object from a **Person** class and an object from a **Dog** class. The two selected bounding boxes obtained after applying the NMS Algorithm 1 are displayed in Figure A.11.

Algorithm 1 (Non-Maximum Suppression algorithm)

Stage 1:

Given a set of all (overlapping and redundant) bounding boxes associated with objects from ℓ classes, perform the following tasks.

- Denote the ℓ class labels by $C_1, \cdots, C_\ell$.
- Denote the coordinates of each bounding box by the vector

$$(b_x, b_y, b_w, b_h, p_c, c_1, \cdots, c_\ell),$$

  where the components are defined as follows.
  - For $i = 1, \cdots, \ell$, the component c_i is equal to 0, unless the class label associated with the box is C_i in which case c_i is equal to 1.
  - p_c is the confidence score associated with the box.
  - The bounding box center is located by the vector (b_x, b_y).
  - b_w and b_h are the bounding box width and height, respectively.
- Set a threshold for the confidence score, say τ_0. This threshold value is the minimum probability at which we believe there is really an object in the bounding box.
- Set a threshold for the IoU, say μ_0. This threshold value is the minimum overlap score at which we believe the two involved bounding boxes are associated with the same object.

Stage 2:

Discard all bounding boxes with component p_c smaller than the threshold τ_0.

Stage 3:

For $i = 1, \cdots, \ell$, perform the following tasks.

- Put all remaining bounding boxes from Stage 2 whose component c_i is equal to 1 into a set, say $\mathscr{B}_i$.
- While there are any remaining bounding boxes in the set $\mathscr{B}_i$, perform the following tasks.
  a) Pick from the set $\mathscr{B}_i$ the bounding box with the largest component p_c and output that as a prediction.
  b) Discard any remaining bounding box in the set $\mathscr{B}_i$ whose IoU with the bounding box output in a) is greater than the threshold μ_0.

1.6 Performance assessment

1.6.1 Introduction

Recall that the goal in object detection is not only to correctly classify objects in an image, but to also find where these objects are located in the image. Hence, a metric for evaluating object detectors must consider in its formulation both the classes and the locations of objects. The assessment of an object detector performance requires the two following data sets.

- A test data set of images along with the ground-truth bounding boxes of all objects to be detected in the images.
- The detections of objects in images from the test data set. Each detection consisting of a bounding box, the corresponding object class and the confidence score.

The mean Average Precision (mAP) is the metric commonly used to evaluate object detector models. The mAP compares all ground-truth bounding boxes to the detected bounding boxes and return a score. The higher the score, the more accurate the model is in its detections. The mAP formula is based on metrics like Precision (P), Recall (R) and Average Precision (AP).

1.6.2 Precision and Recall

To calculate the precision and recall, we need to know the number of detections which can be considered as True Positive (TP), False Positive (FP) and False Negative (FN). These tree types of predictions are similar to those considered when evaluating a binary classification model by a confusion matrix (see Figure A.12). In what follows, we define each of these concepts including the True Negative (TN) one in the context of object detection.

1. A detection is considered as true positive if it satisfies the three following conditions which mean the target object is located at the correct location.
   a) The confidence score of the predicted bounding box is greater than a predefined confidence score threshold, say τ.
   b) The IoU between the predicted bounding box and the ground-truth bounding box is greater than a predefined IoU threshold, say μ.
   c) The class label of the predicted bounding box is the same as the class label of the ground-truth bounding box.
2. A detection is considered as false positive if it does not satisfy at least one of the three conditions a), b) and c) listed above. Figure A.13 shows an illustration of a true positive and a false positive detections.
3. A ground-truth bounding box which is not associated with a detection is considered as a false negative detection.
4. A true negative detection represents the overall area of the image where there are no objects of interest. This area is also referred to as background. Besides, true negative detections are no required in the computations of metrics such as the precision and recall.

Now, assume that in the test data set, there are G ground-truth bounding boxes and that the model outputs N detections from which S are correct ($S \leq G$). The precision

and recall can be expressed as

$$P = \frac{\sum_{n=1}^{S} \mathrm{TP}_n}{\sum_{n=1}^{S} \mathrm{TP}_n + \sum_{n=1}^{N-S} \mathrm{FP}_n} = \frac{\sum_{n=1}^{S} \mathrm{TP}_n}{\text{All detections}},$$

$$R = \frac{\sum_{n=1}^{S} \mathrm{TP}_n}{\sum_{n=1}^{S} \mathrm{TP}_n + \sum_{n=1}^{G-S} \mathrm{FN}_n} = \frac{\sum_{n=1}^{S} \mathrm{TP}_n}{\text{All ground truths}}.$$

It is important to note that all types of detections implicitly depend on the confidence score threshold τ and the IoU threshold μ.

1.6.3 Average precision

Denote the K different confidence scores output by the object detector by τ_k for $k = 1, \cdots, K$, where $\tau_1 < \tau_2 < \cdots < \tau_K$. The precision an recall can be calculated using every confidence score τ_k as threshold. The obtained pairs can be used to draw a precision recall curve like the one provided in Figure A.14. The average precision is defined as the area under the precision recall curve. A good object detector must find all ground-truth objects (high recall) and identify only relevant objects (high precision). The average precision tends to indicate both high precision and high recall, regardless the considered confidence score threshold. In other words, average precision gives a trade off between precision and recall by maximizing the effects of both metrics. Readers can refer to [22] for more details about the computation of average precision. In practice, the average precision is calculated likewise separately for each class.

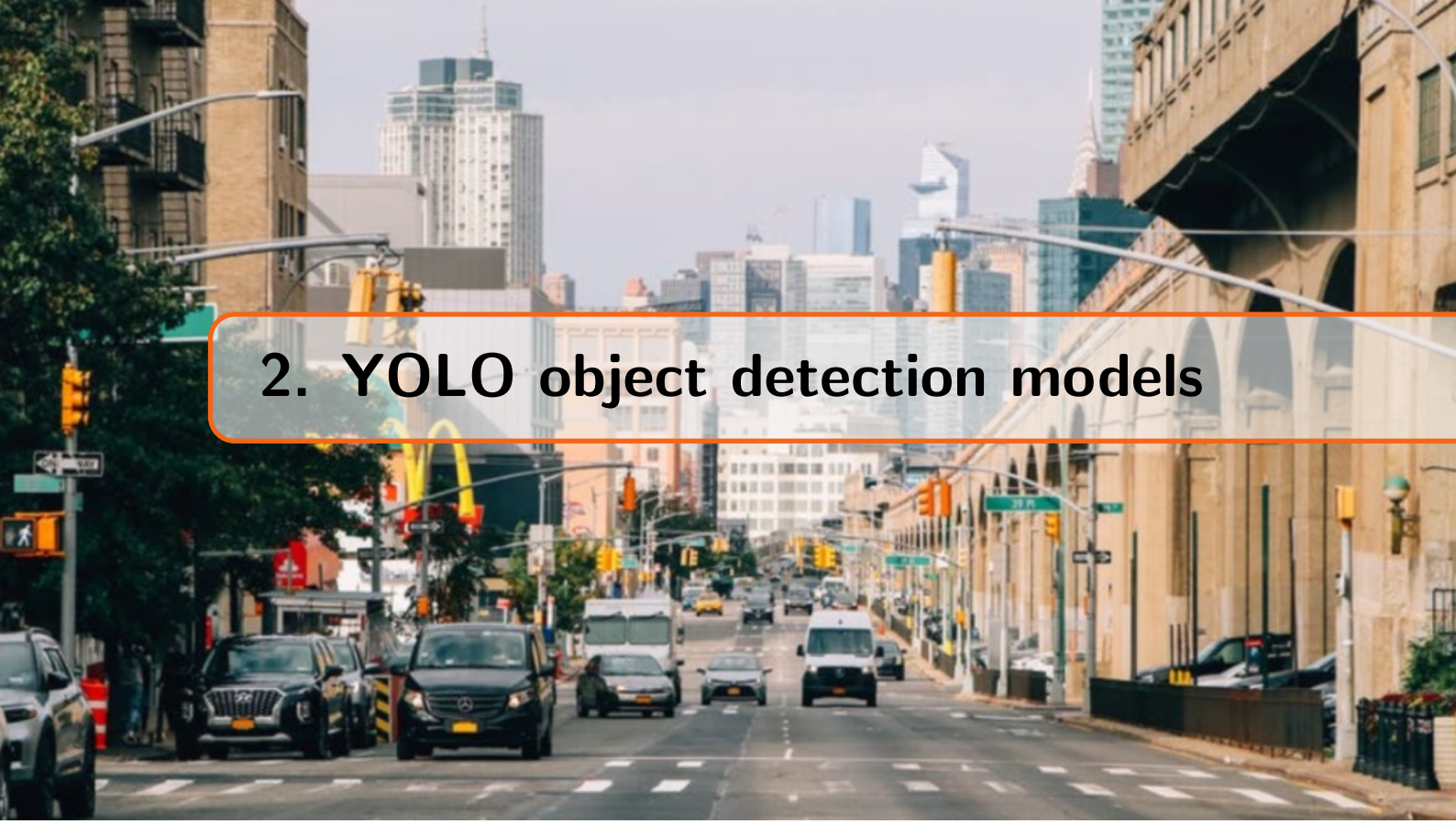
1.6.4 Mean average precision

Denoting the ℓ object class labels by $C_1, \cdots, C_\ell$ and the average precision of the class C_i with respect to the IoU threshold μ by $\mathrm{AP}(C_i, \mu)$, the mean average precision (mAP) with respect to the IoU threshold μ is calculated by the formula

$$\mathrm{mAP}(\mu) = \frac{1}{\ell} \sum_{i=1}^{\ell} \mathrm{AP}(C_i, \mu).$$

Calculating the mean average precision over a range of IoU threshold avoids the ambiguity of picking the optimal IoU threshold. For example, if we denote J different values of the IoU threshold by μ_j, for $j = 1, \cdots, J$ where $\mu_1 < \mu_2 < \cdots < \mu_J$, then the mean average precision over the range $[\mu_1, \mu_J]$ can be calcultaed by the formula

$$\mathrm{mAP}(\mu_1, \cdots, \mu_J) = \frac{1}{J} \sum_{j=1}^{J} \mathrm{mAP}(\mu_j).$$



2. YOLO object detection models

2.1 Introduction

YOLO (You Only Look Once) model is a real-time object detection system which is based on a deep neural network. It provides machines with the capacity to instantaneously detect multiple locations and classes of objects in the images. At the time of writing this report, there are seven main variations of the YOLO object detection model, namely YOLOv1 [26], YOLOv2 [24], YOLOv3 [25], YOLOv4 [3], YOLOv5 (https://github.com/ultralytics/yolov5) and (https://ultralytics.com/yolov5), YOLOv6 (https://github.com/meituan/YOLOv6) and YOLOv7 [31]. The source code and full documentation on training, testing and deployment can be found at

- https://github.com/ultralytics/yolov3 or https://pjreddie.com/darknet/yolo/ for YOLOv3,
- https://github.com/WongKinYiu/PyTorch_YOLOv4 for YOLOv4,
- https://github.com/ultralytics/yolov5 for YOLOv5,
- https://github.com/meituan/YOLOv6 for YOLOv6,
- https://github.com/WongKinYiu/yolov7 for YOLOv7.

Each new version of the YOLO model family includes novel advanced concepts in the previous version to improve the accuracy and the detection time. This section aims to provide an overview of the internal workings common to recent versions of the YOLO model.

2.2 Grid cell

YOLO model divides an image into grids of $S \times S$ cells. For each cell, it tries to predict bounding boxes of objects for which the centers of the ground-truth bounding boxes are located in that cell. Figure B.1 illustrates the identification of the cell responsible for the prediction of an object. It is important to note that the predicted bounding boxes can be larger than the cell responsible for their predictions. Recall that predicting a bounding box means predicting the four parameters to locate the bounding box (the x and y coordinates of the center, the width w and the height h),

a confidence score p and the object class within the bounding box. Denoting the ℓ object class labels by $C_1, \cdots, C_\ell$, the class label $C^* \in \{C_1, \cdots, C_\ell\}$ of the object inside the bounding box is defined by

$$C^* = \underset{C \in \{C_1, \cdots, C_\ell\}}{\operatorname{argmax}} \{q(C)\},$$

where $q(C)$ is the conditional probability of the class C given that there is an object within the bounding box.

At each grid cell, YOLO model uses B anchor boxes (predefined bounding boxes with useful dimensions) as a priori predictions. It is suggested that the B anchor boxes can be chosen as the centroids from the k-means clustering (in which an appropriate metric is used and the number k of clusters is set to be equal to B) performed on the ground-truth bounding boxes associated with all objects in the training data set of images. Anchor boxes picked in this way have the dimensions of the B most common and representative bounding boxes associated with objects in the training data. Figure B.2 shows a set of five different anchor boxes which can result from the k-means clustering.

With the designed bounding boxes in place, the next stage is to correctly pair ground-truth bounding boxes with anchor boxes so that they can be compared. This pairing is done by computing the intersection over union (IoU) between all ground-truth bounding boxes and the B anchor boxes defined in the considered grid cell, and selecting the pairings where the IoU is the highest. An anchor box is responsible for an object if it is paired with a ground-truth bounding box, in which case its confidence score is equal to the IoU between this anchor box and the associated ground-truth bounding box. An anchor box is not responsible for an object if it is not associated with a ground-truth bounding box, in which case its confidence score is equal to 0. Figure B.3 illustrates the notion of ***responsibility*** by showing how an anchor box is assigned to an object (ground-truth bounding box). More formally, the confidence score p of a given bounding box $\mathbf{B}$ is equal to the probability $\Pr(\text{object})$ of the presence of an object in that bounding box multiplied by the IoU between this bounding box and the associated ground-truth bounding box $\mathbf{B}_\mathrm{g}$, that is

$$p = \Pr(\text{object}) \times \operatorname{IoU}\left(\mathbf{B}, \mathbf{B}_\mathrm{g}\right).$$

To summarize, ground-truth bounding boxes are assigned to grid cells by their centers and to anchor boxes by IoU.

With the paired bounding boxes in place, the final stage is to predict the offsets of each anchor box to match the associated ground truth bounding boxes. Figure B.4 shows how a YOLO model represents the offsets. Assume that the considered grid cell is offset from the top left corner of the image by (c_x, c_y). Assume that in that cell, a given anchor box has the width p_w and height p_h. YOLO model predicts five parameters $(t_x, t_y, t_w, t_h, t_o)$ and uses them to place the center of the bounding box predicted from that anchor box at the point (b_x, b_y) and assign to this bounding box the width b_w, the height b_h and the confidence score b_o, where

$$b_x = \sigma(t_x) + c_x, \quad b_y = \sigma(t_y) + c_y,$$

$$b_w = p_w e^{t_w}, \quad b_h = p_h e^{t_h}, \quad b_o = \sigma(t_o)$$

in which $\sigma(\cdot)$ is the sigmoid function.

2.3 Loss function

During training, YOLO model has to take into account errors in the bounding boxes locations and dimensions as well as misclassification errors. It also has to penalize detections of objects where there are not any, namely the false positive detections. YOLO model uses sum of squared errors between the predictions and the ground-truths to calculate loss. The loss function includes the classification loss ($\mathscr{L}_{\text{class}}$), the localization loss ($\mathscr{L}_{\text{box}} = \mathscr{L}_{\text{center}} + \mathscr{L}_{\text{dim}}$) and the confidence score loss $\left(\mathscr{L}_{\text{score}} = \mathscr{L}_{\text{obj}} + \mathscr{L}_{\text{noobj}}\right)$. The final loss adds up all these partial losses explicitly defined below.

2.3.1 Classification loss

If an object is detected in a bounding box, the classification loss is

$$\mathscr{L}_{\text{class}} = \lambda_{\text{class}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{i,j}^{\text{obj}} \sum_{C \in \{C_1, \cdots, C_\ell\}} \left(q_{i,j}(C) - \widehat{q}_{i,j}(C)\right)^2,$$

where

- $\mathbf{1}_{i,j}^{\text{obj}} = 1$ if the j-th anchor box in the cell i is responsible for detecting an object, otherwise 0,
- $q_{i,j}(C) = 1$ if the j-th anchor box in the cell i is responsible for detecting an object from the class C, otherwise 0,
- $\widehat{q}_{i,j}(C)$ is the conditional probability for an object from the class C to be present in the j-th predicted bounding box in the cell i,
- The default value of the penalizing parameter λ_{class} is equal to 1.

2.3.2 Localization loss

1. If an object is detected in a bounding box, the localization loss with respect to the center is

$$\mathscr{L}_{\text{center}} = \lambda_{\text{center}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{i,j}^{\text{obj}} \left[\left(x_{i,j} - \widehat{x}_{i,j}\right)^2 + \left(y_{i,j} - \widehat{y}_{i,j}\right)^2\right],$$

   where
   - $\mathbf{1}_{i,j}^{\text{obj}} = 1$ if the j-th anchor box in the cell i is responsible for detecting an object, otherwise 0,
   - $\left(x_{i,j}, y_{i,j}\right)$ is the center of the ground-truth bounding box associated with the j-th anchor box in the cell i,
   - $\left(\widehat{x}_{i,j}, \widehat{y}_{i,j}\right)$ is the center of the j-th predicted bounding box in the cell i,
   - The default value of the penalizing parameter λ_{center} is equal to 5.
2. If an object is detected in a bounding box, the localization loss with respect to the dimensions is

$$\mathscr{L}_{\text{dim}} = \lambda_{\text{dim}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{i,j}^{\text{obj}} \left[\left(\sqrt{w_{i,j}} - \sqrt{\widehat{w}_{i,j}}\right)^2 + \left(\sqrt{h_{i,j}} - \sqrt{\widehat{h}_{i,j}}\right)^2\right],$$

where

- $\mathbf{1}_{i,j}^{\text{obj}} = 1$ if the j-th anchor box in the cell i is responsible for detecting an object, otherwise 0,
- $w_{i,j}$ and $h_{i,j}$ are respectively the width and height of the ground-truth bounding box associated with the j-th anchor box in the cell i,
- $\widehat{w}_{i,j}$ and $\widehat{h}_{i,j}$ are respectively the width and height of the j-th predicted bounding box in the cell i,
- The default value of the penalizing parameter λ_{dim} is equal to 5.

2.3.3 Confidence score loss

1. If an object is detected in a bounding box, the confidence score loss is

$$\mathscr{L}_{\text{obj}} = \lambda_{\text{obj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{i,j}^{\text{obj}} \left(p_{i,j} - \widehat{p}_{i,j}\right)^2,$$

where

- $\mathbf{1}_{i,j}^{\text{obj}} = 1$ if the j-th anchor box in the cell i is responsible for detecting an object, otherwise 0,
- $\widehat{p}_{i,j} = 1$ if the j-th anchor box in the cell i is responsible for detecting an object, otherwise 0,
- $\widehat{p}_{i,j}$ is the confidence score of the j-th predicted bounding box in the cell i,
- The default value of the penalizing parameter λ_{obj} is equal to 1.

2. If an object is not detected in a bounding box, the confidence score loss is

$$\mathscr{L}_{\text{noobj}} = \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{i,j}^{\text{noobj}} \left(p_{i,j} - \widehat{p}_{i,j}\right)^2$$

where

- $\mathbf{1}_{i,j}^{\text{noobj}} = 1$ if the j-th anchor box in the cell i is not responsible for detecting an object, otherwise 0,
- The default value of the penalizing parameter λ_{noobj} is equal to 0.5.

2.4 Internal workings

Recall that a YOLO model divides an image into a $S \times S$ grid cells and that B anchor boxes are assigned to each grid cell for predictions. Each predicted bounding box includes the two coordinates (x, y) for the center, the width w and height h, the confidence score p and the ℓ class conditional probabilities $q(C_1), \cdots, q(C_\ell)$. YOLO object detector models detection as a regression problem in which the features extracted from the training images by a deep convolutional neural network model together with the anchor boxes from all grid cells are independent variables and all ground-truth bounding boxes along with the backgrounds are dependent variables. Precise that feature extraction and the nonlinear regression model are optimized simultaneously as a single process. In addition, this regression model is also designed as a deep neural network model.

In the YOLO model, the prediction associated with an entire image is encoded as a vector Ω of $S \times S \times B \times (5+\ell)$ dimensions, namely

$$\Omega = \left(\Omega_i,\ i = 0, 1, 2, \cdots, S^2\right),$$

where Ω_i is the vector of $B \times (5+\ell)$ dimensions defined by

$$\Omega_i = \left(\Omega_{i,j},\ j = 0, 1, 2, \cdots, B\right)$$

in which $\Omega_{i,j}$ is the following vector of $(5+\ell)$ dimensions

$$\Omega_{i,j} = \left(x_{i,j}, y_{i,j}, w_{i,j}, h_{i,j}, p_{i,j}, q_{i,j}(C_1), \cdots, q_{i,j}(C_\ell)\right).$$

Figure B.5 show a summary of the internal workings of the YOLO model in action on an image. During the inference, object detectors can make duplicated detections for the same object. To fix this, YOLO model makes use of the non-maximum suppression technique to remove weaker detections that have too much overlap with stronger detections.

3. Images of traffic scene objects

3.1 Introduction

In recent years, many object detection algorithms have emerged. The pre-trained deep learning models in computer vision available for free are tailored to detect specific classes of objects in daily life. In the case where these ready-to-use models are not suitable for an application, a custom object detector is required. The two following tasks must be completed to build a custom object detector for the detection of some target classes of objects.

1. Collect several images in which at least one of the considered classes of objects is represented. It is recommended that in the set of all collected images, each class of interest is represented by several hundred of objects.
2. Annotate all target objects present in each collected image. One should make sure that the selected tool exploited to annotate images allows the extraction of annotations in a format supported by the implemented object detection algorithm to use.

This chapter provides some processes and technical tools which can be used to accomplish both tasks in the particular case where the considered classes of objects can be found in traffic scenes. After a brief description of the considered classes of objects, we indicate the sources of images which form the training data set. Then, we end with a succinct presentation of some annotation tools and formats.

3.2 Classes of traffic scene objects

One of the primary goals in computer vision is the understanding of visual scenes. Scene understanding involves numerous tasks including the recognition and the localization of all objects which are present therein as well as the characterization of relationships between these objects. Since recognition and localization often rely on the results from detection, the ability to detect objects of interest effectively plays a crucial role in scene understanding or perception. In particular, understanding traffic scene requires to extract in real-time accurate road environment information. This

section describes 35 important classes of objects which can be encountered in road environment. The considered classes of traffic scene objects are named as follows. **tree (group)**, **airplane**, **truck**, **car**, **gas station**, **tree**, **electric pole**, **helicopter**, **person**, **person (group)**, **tenement**, **bench (group)**, **street light**, **traffic sign**, **fence**, **traffic light**, **house (group)**, **tunnel**, **bus**, **house**, **bird**, **parking meter**, **tunnel entrance**, **bird (group)**, **bench**, **special building**, **car (group)**, **fire hydrant**, **motorcycle**, **scooter**, **stop sign**, **train**, **dog**, **bicycle**, **boat**. Samples of objects from all these classes can be found in Figures C.1–C.38.

3.3 Collecting images of traffic scene objects

The first step towards creating a good custom computer vision model for a specific application is collecting the training data set. This can be a challenging task as a deep learning model requires to collect huge amount of images containing various objects of interest to train its algorithm. Indeed, the pre-trained models available for free to detect some classes of objects in daily life are trained on existing large benchmark data sets such as Common Objects is Context (COCO) [18] available at https://cocodataset.org/#home, ImageNet [7] available at https://www.image-net.org/ and PASCAL Visual Object Classes (PASCAL VOC) [8, 9] available at http://host.robots.ox.ac.uk/pascal/VOC/. Here, PASCAL stands for pattern analysis, statistical modeling and computational learning. In addition, these pre-trained object detectors are based on the classical algorithms, namely the Mask-RCNN (see, [12] and https://github.com/matterport/Mask_RCNN), the YOLO (see, https://github.com/ultralytics/yolov5 and https://github.com/WongKinYiu/yolov7) and the MobileNet (see, [13] and https://keras.io/api/applications/#mobilenet).

In this section, we explain some techniques used for collecting image data of traffic scene objects. These techniques include searching through public open data sets and scraping the web. All data are extracted and combined from different sources by using as key words the 35 object class names mentioned in the previous section. Note that we make use of the fiftyone python library https://pypi.org/project/fiftyone/, as well as the Download All Images tool https://download-all-images.mobilefirst.me/ for bulk downloads. The sources of training data sets of images are listed below.

1. COCO data set by means of the FiftyOne tool.
2. Google Open Images data set https://storage.googleapis.com/openimages/web/index.html by means of the FiftyOne tool.
3. Web scraping to search and download images either manually or by means of the Download All Images tool.

We end this section with the following information regarding the Download All Images and FiftyOne tools as well as the Google Open Images, the COCO and the PASCAL VOC data sets.

- Download All Images tool is an extension on Google Chrome to download a bunch of images at once.
- FiftyOne tool https://voxel51.com/docs/fiftyone/ is an open source tool which includes the functionalities to access, to download and to visualize the COCO and Google Open Images data sets.

- Google Open Images is a very large data set developed by Google which contains several million images for several hundreds of different object classes.
- COCO is a large data set developed by Microsoft which contains several hundred thousand images for 80 different object classes, namely **person, bicycle, car, motorcycle, airplane, bus, train, truck, boat, traffic light, fire hydrant, stop sign, parking meter, bench, bird, cat, dog, horse, sheep, cow, elephant, bear, zebra, giraffe, backpack, umbrella, handbag, tie, suitcase, frisbee, skis,snowboard, sports ball, kite, baseball bat, baseball glove, skateboard, surfboard, tennis racket, bottle, wine glass, cup, fork, knife, spoon, bowl, banana, apple, sandwich, orange, broccoli, carrot, hot dog, pizza, donut, cake, chair, couch, potted plant, bed, dining table, toilet, tv, laptop, mouse, remote, keyboard, cell phone, microwave, oven, toaster, sink, refrigerator, book, clock, vase, scissors, teddy bear, hair drier, toothbrush.**
- PASCAL VOC is a medium data set collected from flickr https://www.flickr.com/ which contains several thousand images for 20 different object classes, namely **aeroplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, dining table, potted plant, sofa, tvmonitor, bird, cat, cow, dog, horse, sheep, person.**

3.4 Labeling classes of traffic scene objects

Image annotation for object detection refers to the practice of attaching a file of labels to an image. Each label consists of the name or index associated with an object class of interest along with the coordinates of a bounding box which specifies the location of that object in the image. Recall that bounding box is a rectangular frame used to surround a target object.

3.4.1 Annotation tools

Data labeling is an essential step in a supervised machine learning task as the quality of training data generally determines how well our model performs. This conclusion also holds true for the annotation files associated with image files which form the training data set. If one shows to a child a bus and say its a truck, the next time the child sees a bus, it is very likely that he classifies it as a truck. As a machine learning model learns in a similar way, by looking at examples, the result of the model depends on the labels feed in during its training phase. Labeling images usually involves to perform delicate manual tasks in an image annotation tool. Here is a list of non-exhaustive tools that can be used to annotate images.

- LabelImg: It is available either at https://pypi.org/project/labelImg/ or https://github.com/heartexlabs/labelImg.
- LabelStudio: It is available either at https://github.com/heartexlabs/label-studio or https://labelstud.io/guide/index.html#Quick-start.
- ImgLab: It is available either at https://imglab.in/ or https://github.com/imglab-io/imglab-py or https://github.com/NaturalIntelligence/imglab.
- RectLabel: It is available at https://rectlabel.com/.
- Scalable: It is available either at https://scalabel.ai/ or https://github.com/scalabel/scalabel.

- Make-Sense: It is available either at https://www.makesense.ai/ or https://github.com/SkalskiP/make-sense.
- CVAT (Computer Vision Annotation Tool): It is available either at https://www.cvat.ai/ or https://github.com/opencv/cvat.
- Roboflow: It is available at https://roboflow.com/.
- V7: It is available at https://www.v7labs.com/.

3.4.2 Annotation formats

For object detection tasks, there is no single standard format for storing data (object class names and bounding box coordinates) in annotation files. The most commonly used formats include COCO JSON, TensorFlow Object Detection CSV, PASCAL VOC XML and YOLO TXT. These formats determine the structure of data in annotation files as each one uses its specific representation of coordinates associated with bounding boxes. When annotating image data set, the format of annotation files must be compatible with the computer vision algorithm to be used. For example, classical object detection algorithms such as SSD, RCNN, Fast RCNN and Faster RCNN use the PASCAL VOC XML format for annotation files whereas recent versions of YOLO algorithms for real-time object detection use the YOLO TXT format for annotation files. Besides, the formats COCO JSON and Tensorflow Object Detection CSV for annotation files are required when building these standard object detection models on the platforms Detectron2 (https://github.com/facebookresearch/detectron2) and TensorFlow (https://github.com/tensorflow/models/tree/master/research/object_detection), respectively.

In previous sections, the following tasks have been performed with the aim to build a custom object detector for traffic scene objects.

1. Specification of the class labels of all traffic scene objects to detect.
2. Collection of a moderate number of images containing various traffic scene objects to annotate.
3. Presentation of some image annotation tools in which the collected images can be uploaded for labeling.

Based on the arguments stated in the previous paragraph, the next step consists to export the annotations in a convenient or preferred format. When working with deep learning models in computer vision, it is important to be familiar with some of the popular annotation formats and know how to convert them from one format to another for more flexibility in use. A platform like Roboflow (https://roboflow.com/) has a universal annotation conversion tool available at https://roboflow.com/formats that allows users to upload and convert annotations from one format to another without having to write conversion scripts.

Recall that each annotation file contains information about the names and locations of objects on the given image file. The organization of all these informations in annotation files from some different formats is given below. As an illustration, we consider the sample of six images provided in Section C.36 (see, Figure C.39, Figure C.40, Figure C.41, Figure C.42, Figure C.43 and Figure C.44) in which we used the Roboflow annotation tool to draw bounding boxes around 4 classes of objects, namely **bicycle**, **bus**, **car** and **motorcycle**. The annotations are exported in different formats

and displayed in the subsequent sections. Let us note that the width and height of images may not match with the values shown in annotation files due to cropping. The data sets of image files containing traffic scene objects with the corresponding annotation files in the YOLO format can be downloaded at https://drive.google.com/drive/folders/1RZRnfSMzdnQuM6r-3iCHenl8c0dN3X92?usp=sharing for further studies. The next chapter includes an exploratory data analysis of the annotated data set along with the modeling process by means of the YOLO algorithm in order to get a custom object detector for traffic scene objects.

TensorFlow Object Detection CSV format

Annotations exported in TensorFlow Object Detection CSV format is a single file of comma separated values (csv) having the fields called **filename**, **width**, **height**, **class**, **xmin**, **ymin**, **xmax** and **ymax**. An example of such a file is displayed in the Listing 3.1, where the annotated image files can be found in Section C.36. In this file, one row contains only the information about one object in an image. The fields of this csv file are explicitly described as follows.

- The field **filename** refers to the name of an image file.
- The fields **width** and **height** represent the image width and height, respectively.
- The field **class** stands for the class name of an object.
- The fields **xmin** and **ymin** correspond respectively to the **x** and **y** coordinates of the top left corner of a bounding box.
- The fields **xmax** and **ymax** correspond respectively to the **x** and **y** coordinates of the bottom right corner of a bounding box.

Listing 3.1: TensorFlow Object Detection CSV file

```
filename      width height class       xmin ymin xmax ymax
images1.jpg   640   640    motorcycle  22   190  238  541
images1.jpg   640   640    car         195  153  614  530
images2.jpg   640   640    car         30   168  352  518
images2.jpg   640   640    car         299  110  600  400
images3.jpg   640   640    bus         28   0    236  628
images3.jpg   640   640    bus         240  24   489  588
images4.jpg   640   640    bus         37   167  286  467
images4.jpg   640   640    bus         288  160  511  470
images5.jpg   640   640    bus         0    122  260  518
images5.jpg   640   640    bus         256  145  636  522
images6.jpg   640   640    bicycle     62   126  250  502
images6.jpg   640   640    car         168  238  539  426
```

YOLO TXT format

Annotations exported in YOLO TXT format consist of multiple text files with the same names as the image files plus a complementary YAML file. Each text file contains annotations for all objects in the corresponding image file. In a text file, each line contains only information about one object. These information are structured in the following form:

<object-class-index> <center-x> <center-y> <width> <height>.

Examples of such text files are displayed in the Listing 3.2-3.7, where the annotated image files can be found in Section C.36. The fields of each text file are explicitly described as follows.

- The object class names are gathered into a list which is provided as value of the field called **names** in the additional YAML file as the one displayed in the Listing 3.8. The value **<object-class-index>** corresponds to the order of elements in that list, the index value 0 being for the first element, the index value 1 for the second element, and so on.
- The values **<center-x>** and **<center-y>** refer respectively to the **x** and **y** coordinates associated with the center of a bounding box. Here, the **x** coordinate is divided by the image width whereas the **y** coordinate is divided by the image height.
- The values **<width>** and **<height>** represent the width and height of a bounding box, respectively. Here the width is divided by the image width whereas the height is divided by the image height.

It is important to note that in the YOLO TXT format, the bounding box width and height along with the coordinates of its center are normalized between 0 and 1 as percentage of image dimensions.

Listing 3.2: YOLO TXT file: images1.txt

```
3 0.20234375 0.57109375 0.3375     0.54765625
2 0.63203125 0.5328125  0.65546875 0.5890625
```

Listing 3.3: YOLO TXT file: images2.txt

```
2 0.2984375 0.5359375 0.503125  0.54765625
2 0.7015625 0.3984375 0.4703125 0.45234375
```

Listing 3.4: YOLO TXT file: images3.txt

```
1 0.20625    0.490625 0.32421875 0.98125
1 0.56953125 0.478125 0.38984375 0.88046875
```

Listing 3.5: YOLO TXT file: images4.txt

```
1 0.2515625  0.49453125 0.3890625  0.46875
1 0.62421875 0.4921875  0.34765625 0.484375
```

Listing 3.6: YOLO TXT file of images5.txt

```
1 0.203125 0.5        0.40625    0.61875
1 0.696875 0.52109375 0.59296875 0.5890625
```

Listing 3.7: YOLO TXT file: images6.txt

```
0 0.24296875 0.48984375 0.29375    0.5875
2 0.55234375 0.51796875 0.57890625 0.29375
```

Listing 3.8: YOLO TXT file: data.yaml

```
train: ../train/images
val: ../valid/images
test: ../test/images

nc: 4
names: ['bicycle', 'bus', 'car', 'motorcycle']
```

PASCAL VOC XML format

Annotations exported in PASCAL VOC XML format consist of multiple xml files having the same names as the image files. Each xml file contains annotations for all objects in the corresponding image file. Examples of such xml files are displayed in the Listing 3.9-3.10, where the annotated image files can be found in Section C.36. Some components or tags in these xml files are explicitly described as follows.

- The tag **folder** is the directory that contains the considered image file.
- The tag **filename** is the name of the considered image file.
- The tag **source** is the database name (if there is any) in which the considered image file is stored.
- The tag **size** contains the image height and width expressed in terms of pixels as well as the image depth indicating the number of channels. For a color image the depth is equal to 3 whereas for a black and white image the depth is equal to 1.
- The tag **object** contains only information about one object in the considered image. These informations consist of the class name of an object along with the coordinates of the top left corner (xmin, ymin) and the bottom right corner (xmax, ymax) of the bounding box surrounding that object. Let us precise that this tag is repeated as many times as there are objects in the image.

Listing 3.9: PASCAL VOC XML file: images1.xml

```
<annotation>
    <folder></folder>
    <filename>images1.jpg</filename>
    <path>images1.jpg</path>
    <source>
        <database>roboflow.ai</database>
    </source>
    <size>
        <width>640</width>
        <height>640</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>motorcycle</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <occluded>0</occluded>
        <bndbox>
            <xmin>23</xmin>
            <xmax>239</xmax>
            <ymin>191</ymin>
            <ymax>542</ymax>
        </bndbox>
    </object>
    <object>
        <name>car</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <occluded>0</occluded>
        <bndbox>
            <xmin>196</xmin>
            <xmax>615</xmax>
            <ymin>154</ymin>
            <ymax>531</ymax>
        </bndbox>
    </object>
</annotation>
```

Listing 3.10: PASCAL VOC XML file: images6.xml

```
<annotation>
    <folder></folder>
    <filename>images6.jpg</filename>
    <path>images6.jpg</path>
    <source>
        <database>roboflow.ai</database>
    </source>
    <size>
        <width>640</width>
        <height>640</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>bicycle</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <occluded>0</occluded>
        <bndbox>
            <xmin>63</xmin>
            <xmax>251</xmax>
            <ymin>127</ymin>
            <ymax>503</ymax>
        </bndbox>
    </object>
    <object>
        <name>car</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <occluded>0</occluded>
        <bndbox>
            <xmin>169</xmin>
            <xmax>540</xmax>
            <ymin>239</ymin>
            <ymax>427</ymax>
        </bndbox>
    </object>
</annotation>
```

COCO JSON format

Annotations in COCO JSON format are exported as a single json file having the global structure like the one displayed in the Listing 3.11.

Listing 3.11: COCO JSON file (Global structure)

```
[{
  "info": {...},
  "licenses": [...],
  "categories": [...],
  "images": [...],
  "annotations": [...]
}]
```

An example of such a json file is partitioned and displayed through the Listing 3.12-3.16, where the annotated image files can be found in Section C.36. In this example, the fields called **info**, **licenses** and **categories** can be found in the Listing 3.12. Besides, the field called **images** is given in the Listing 3.13 whereas the field called **annotations** has its content distributed in the Listing 3.14-3.16. All these fields are explicitly described as follows.

- The field **info** is a dictionary which contains metadata about the data set of images.
- The field **licenses** is a list which contains all types of licenses associated with the data set of images. Let us precise that information about distinct licenses are put in separated dictionaries.
- The field **categories** contains list of dictionaries, with one dictionary for each object class. It assigns a category **id** to each object class name. These **id** are used in the **annotations** field instead of object class names.
- The field **images** contains a list of dictionaries, with one dictionary for each image in the data set. This field includes some metadata such the image width and height. It also assigns an image **id** to each image name. These **id** are used in the **annotations** field.
- The field **annotations** contains a list of dictionaries, with one dictionary for each object. This field assigns a unique annotation **id** to each object and indicates the image **id** and the category **id** to which the object relates. The coordinates of the bounding box associated with the object are indicated in the sub field called **bbox** in the form [xmin, ymin, width, height], where (xmin, ymin) locates the top left corner of the bounding box having the values width and height as dimensions. Area of the bounding box is given in the sub field called **area**.

Listing 3.12: COCO JSON file (Part 1)

```
[{
        "info":
        {
            "year": "2022",
            "version": "1",
            "description": "Exported from roboflow.ai",
            "contributor": "",
            "url": "https://public.roboflow.ai/object-detection/
                undefined",
            "date_created": "2022-10-30T17:33:33+00:00"
        },
        "licenses":
        [{
            "id": 1,
            "url": "https://creativecommons.org/licenses/by/4.0/",
            "name": "CC BY 4.0"
        }],
        "categories":
        [{
            "id": 0,
            "name": "car-bus-bicycle-motorcycle",
            "supercategory": "none"
        },{
            "id": 1,
            "name": "bicycle",
            "supercategory": "car-bus-bicycle-motorcycle"
        },{
            "id": 2,
            "name": "bus",
            "supercategory": "car-bus-bicycle-motorcycle"
        },{
            "id": 3,
            "name": "car",
            "supercategory": "car-bus-bicycle-motorcycle"
        },{
            "id": 4,
            "name": "motorcycle",
            "supercategory": "car-bus-bicycle-motorcycle"
        }],
```

Listing 3.13: COCO JSON file (Part 2)

```
    "images":
    [{
        "id": 0,
        "license": 1,
        "file_name": "images5.jpg",
        "height": 640,
        "width": 640,
        "date_captured": "2022-10-30T17:33:33+00:00"
    },{
        "id": 1,
        "license": 1,
        "file_name": "images4.jpg",
        "height": 640,
        "width": 640,
        "date_captured": "2022-10-30T17:33:33+00:00"
    },{
        "id": 2,
        "license": 1,
        "file_name": "images2.jpg",
        "height": 640,
        "width": 640,
        "date_captured": "2022-10-30T17:33:33+00:00"
    },{
        "id": 3,
        "license": 1,
        "file_name": "images3.jpg",
        "height": 640,
        "width": 640,
        "date_captured": "2022-10-30T17:33:33+00:00"
    },{
        "id": 4,
        "license": 1,
        "file_name": "images1.jpg",
        "height": 640,
        "width": 640,
        "date_captured": "2022-10-30T17:33:33+00:00"
    },{
        "id": 5,
        "license": 1,
        "file_name": "images6.jpg",
        "height": 640,
        "width": 640,
        "date_captured": "2022-10-30T17:33:33+00:00"
    }],
```

Listing 3.14: COCO JSON file (Part 3)

```
    "annotations":
    [{
        "id": 0,
        "image_id": 0,
        "category_id": 2,
        "bbox": [0,122,260,396],
        "area": 102960,
        "segmentation": [],
        "iscrowd": 0
    },{
        "id": 1,
        "image_id": 0,
        "category_id": 2,
        "bbox": [256,145,379.5,377],
        "area": 143071.5,
        "segmentation": [],
        "iscrowd": 0
    },{
        "id": 2,
        "image_id": 1,
        "category_id": 2,
        "bbox": [37,167,249,300],
        "area": 74700,
        "segmentation": [],
        "iscrowd": 0
    },{
        "id": 3,
        "image_id": 1,
        "category_id": 2,
        "bbox": [288,160,222.5,310],
        "area": 68975,
        "segmentation": [],
        "iscrowd": 0
    },
```

Listing 3.15: COCO JSON file (Part 4)

```
        {
            "id": 4,
            "image_id": 2,
            "category_id": 3,
            "bbox": [30,168,322,350.5],
            "area": 112861,
            "segmentation": [],
            "iscrowd": 0
        },{
            "id": 5,
            "image_id": 2,
            "category_id": 3,
            "bbox": [299,110,301,289.5],
            "area": 87139.5,
            "segmentation": [],
            "iscrowd": 0
        },{
            "id": 6,
            "image_id": 3,
            "category_id": 2,
            "bbox": [28,0,207.5,628],
            "area": 130310,
            "segmentation": [],
            "iscrowd": 0
        },{
            "id": 7,
            "image_id": 3,
            "category_id": 2,
            "bbox": [240,24,249.5,563.5],
            "area": 140593.25,
            "segmentation": [],
            "iscrowd": 0
        },
```

Listing 3.16: COCO JSON file (Part 5)

```
        {
            "id": 8,
            "image_id": 4,
            "category_id": 4,
            "bbox": [22,190,216,350.5],
            "area": 75708,
            "segmentation": [],
            "iscrowd": 0
        },{
            "id": 9,
            "image_id": 4,
            "category_id": 3,
            "bbox": [195,153,419.5,377],
            "area": 158151.5,
            "segmentation": [],
            "iscrowd": 0
        },{
            "id": 10,
            "image_id": 5,
            "category_id": 1,
            "bbox": [62,126,188,376],
            "area": 70688,
            "segmentation": [],
            "iscrowd": 0
        },{
            "id": 11,
            "image_id": 5,
            "category_id": 3,
            "bbox": [168,238,370.5,188],
            "area": 69654,
            "segmentation": [],
            "iscrowd": 0
        }]
    }]
```

4. Detector for traffic scene objects

4.1 Introduction

Object detection is a branch of computer vision that identifies and locates objects in an image or video. Complex tasks like self-driving cars make use of object detection for understanding different scenarios and making decisions based on them. Ultralytics provides a simple and powerful approach for training custom object detection models using the YOLOv5 architetures. Creating a custom model to detect objects of interest is a process of collecting and organizing images, labeling objects of interest, training a model and using the trained model to make inferences on new images. The purpose of this chapter is to describe all steps involved in the training process of a model in the YOLOv5 family while focusing on the practical use case of building a custom model to detect traffic scene objects. This chapter is organized as follows. We start with a brief presentation of models in the YOLOv5 family in Section 4.2. Then, we discuss the essential steps to train a custom YOLOv5 model in Section 4.3. Next, Section 4.4 explains how to evaluate performances of the trained model. Finally, Section 4.5 indicates how to make inferences or predictions on image or video files with the trained model.

4.2 YOLOv5 family

YOLOv5 is a collection of different object detection model architectures. These models can be classified in five groups according to their sizes as follows.

1. Nano models (YOLOv5n and YOLOv5n6).
2. Small models (YOLOv5s and YOLOv5s6).
3. Medium models (YOLOv5m and YOLOv5m6).
4. Large models (YOLOv5l and YOLOv5l6).
5. Extra large models (YOLOv5x and YOLOv5x6).

All models in the YOLOv5 family are pretrained on the MS COCO dataset (https://cocodataset.org/#home) and can be used to detect 80 classes of objects on image or video files. Figure D.1 shows an overview of models available in YOLOv5 family.

The following informations are provided for each model:

- the input image size (in pixels),
- the total number of weight parameters (in million),
- the inference speed on CPU and GPU (in milliseconds),
- the mean average precision (mAP) associated with an intersection over union (IoU) of 0.5 denoted by mAP 0.5,
- the mean average precision (mAP) associated with intersection over union (IoU) varying from 0.5 to 0.95 by a step of 0.05 denoted by mAP 0.5 : 0.95.

4.3 Custom YOLOv5 learning

This section explains how to train a model from the YOLOv5 family to recognize custom objects on images.

4.3.1 YOLOv5 installation

The starting point is to set up the YOLOv5 programming environment to run object detection training and inference commands. The following lines of codes can be used to automate this installation in an environment including recent versions of Python and PyTorch machine learning framework.

```
git clone https://github.com/ultralytics/yolov5  # clone
cd yolov5
pip install -r requirements.txt  # install
```

These three lines of codes can be explained as follows.

- Command in the first line is used to clone the repository of yolov5.
- Command in the second line is used to enter into root directory of the cloned repository.
- Command in the third line is used to install the required packages from the cloned repository root directory.

4.3.2 Dataset preparation

The first step is to create a data set. YOLOv5 must be trained on labeled data in order to learn classes of objects in that data. Roboflow (https://roboflow.com/) can be used to label, prepare and host a custom data automatically in YOLO format. Alternatively, one can manually prepare a data set by performing the following tasks.

1. Annotate images and export annotations in YOLO format.
2. Create a folder called **datasets** in the working directory containing the **yolov5** folder. Precise that the **yolov5** folder is automatically created when setting up the YOLOv5 programming environment by the above mentioned line of codes.
3. Create a new folder in the **datasets** directory and assign it the project name, say **traffic**.
4. Create two directories called **images** and **labels** in the **traffic** folder.
5. Create two directories called **train** and **val** in the **images** folder. Then, put all images for training in the path **traffic/images/train** and all images for validation in the path **traffic/images/val**.

6. Create two directories called **train** and **val** in the **labels** folder. Then, put all annotations associated with training images in the path **traffic/labels/train** and all annotations associated with validation images in the path **traffic/labels/val**.
7. In the **datasets/traffic** directory, create a .yaml file, say **traffic.yaml** (see Listing 4.1) that defines the locations of the directories containing data sets associated with the project. It also specifies the total number of classes along with the names of each class. Note that in the .yaml file, the paths to annotations (labels) files are not specified since YOLOv5 automatically locates labels associated with the images by following the paths.

   ```
   # train and val data
   train: ../datasets/traffic/labels/train/
   val: ../datasets/traffic/labels/val/
   test: ../datasets/traffic/labels/test/ # test images (optional)
   ```

   Consequently, the **images** and **labels** folders must be defined in the design data set directories, unless images and corresponding annotations are located into the same paths for training and validation data provided in the .yaml file.
8. Optionally, one can create a test folder in the **images** and **labels** directories. Then put all images for test in the path **traffic/images/test** and all annotations associated with the test images in the path **traffic/labels/test**. This will automatically trigger the assessment of model performance on test data when the training process ends.

Listing 4.1: traffic.yaml

```
# train and val data
train: ../datasets/traffic/images/train/
val: ../datasets/traffic/images/val/
test: ../datasets/traffic/images/test/ # test images (optional)

# number of classes
nc: 35

# class names
names: ["tree (group)", "airplane", "truck", "car", "gas station",
"tree", "electric pole", "helicopter", "person", "person (group)",
"tenement", "bench (group)", "street light", "traffic sign",
"fence", "traffic light", "house (group)", "tunnel", "bus",
"house", "bird", "parking meter", "tunnel entrance",
"bird (group)", "bench", "special building", "car (group)",
"fire hydrant", "motorcycle", "scooter", "stop sign",
"train", "dog", "bicycle", "boat"]
```

4.3.3 Data set exploration

This optional step is intended to have an overview on the statistics concerning images and annotations associated with training and validation data. The data set containing images of traffic scene objects along with annotations in YOLO format and .yaml file can be downloaded at https://drive.google.com/drive/folders/1RZRnfSMzdnQuM6r-3iCHenl8c0dN3X92?usp=sharing.

Overall data set

Table D.1 contains some basic statistics associated with the object distribution per image and class on all data set. The overall data set contains 51312 labels (annotations) across 35 classes on 8550 images. The number of objects per class ranges between 174 and 7920 with an average of 1466. The frequencies of different classes of objects can be visualized in Figure D.2 to identify the most frequent object classes to the less frequent ones. Similarly, the number of objects per image ranges between 1 and 55 with an average of 6. The frequencies of images containing a specific number of objects can be visualized in Figure D.3 to have an overview of the most dense images to the less dense ones.

Training data set

Table D.2 contains some basic statistics associated with the object distribution per image and class on the training data set. The training data set contains 39680 labels (annotations) across 35 classes on 6800 images. The number of objects per class ranges between 110 and 5687 with an average of 1133.714. The frequencies of different classes of objects can be visualized in Figure D.4 to identify the most frequent object classes to the less frequent ones. Similarly, the number of objects per image ranges between 1 and 55 with an average of 5.835. The frequencies of images containing a specific number of objects can be visualized in Figure D.5 to have an overview of the most dense images to the less dense ones.

Validation data set

Table D.3 contains some basic statistics associated with the object distribution per image and class on the validation data set. The validation data set contains 11632 labels (annotations) across 35 classes on 1750 images. The number of objects per class ranges between 39 and 2233 with an average of 332.343. The frequencies of different classes of objects can be visualized in Figure D.6 to identify the most frequent object classes to the less frequent ones. Similarly, the number of objects per image ranges between 1 and 53 with an average of 6.647. The frequencies of images containing a specific number of objects can be visualized in Figure D.7 to have an overview of the most dense images to the less dense ones.

4.3.4 Custom YOLOv5 selection

The second step is to select a pretrained YOLOv5 model to start training from. Here, we select the YOLOv5s, the second smallest and fast model available in the YOLOv5 family. Depending on the computer resources and the desired level of performance, one can choose another member in the YOLOv5 family.

4.3.5 Custom YOLOv5 training

The third step is to train a YOLOv5s model on traffic data set by using the following command after getting into the **yolov5** folder which contains the **train.py** function:

```
$ python train.py --img 640 --batch 16 --epochs 250
--data ../datasets/traffic/traffic.yaml --weights yolov5s.pt
```

where the used standard model parameters are:

- **img**: to indicate the input image size. The input images will be automatically resized to the value assigned to this parameter before feeding the network defined by the model.
- **batch**: to indicate the number of images fed at once into the network defined by the model. This value must be set according to the memory available in the computer.
- **epochs**: to define the number of times to train the model (update the model weights) on the full data set.
- **data**: to specify location of the defined .yaml file.
- **weights**: to specify location of weights associated with the selected pretrained YOLOv5 model.

Note that when executing the above command for the first time, the specified weights, here **yolov5s.pt** are auto-downloaded from the latest YOLOv5 release at https://github.com/ultralytics/yolov5/releases.

4.4 Custom YOLOv5 performance

Each time a train command similar to the one provided above is executed for an experiment of learning a custom object detection, results are saved by default to a new incrementing directory as follows. All results associated with first training experiment are saved by default to the path **yolov5/runs/train/exp** whereas results associated with a subsequent k-th training experiment are saved by default to the path **yolov5/runs/train/expk**. These directories contain training and validation statistics as well as some metrics including precision, recall and confusion matrix. Performance metrics on the validation data of the trained model associated with the traffic scene objects are displayed in Figure D.8 for precision and recall, in Figure D.9 for precision and recall per class of objects, and in Figure D.10 for confusion matrix. These results show that the trained object detector performs pretty good at detecting objects from different classes of interest. However, one can improve these performances by retraining the model on large among of data in which the numbers of objects from different classes are balanced.

4.5 Custom YOLOv5 inference

After training is completed, model weights associated with the training experiment are saved by default to new incrementing directory as follows.

1. Weights associated with the first training experiment are saved by default to the path **yolov5/runs/train/exp/weights**.
2. Weights associated with a subsequent **k**-th training experiment are saved to the path **yolov5/runs/train/expk/weights**.

These directories contain weights obtained at the last epoch called **last.pt** and those yielding the best results on validation data called **best.pt**.

To run inference concerned with the estimated model from the first experiment, one can use the following command after getting into the **yolov5** folder which contains the **detect.py** function:

```
python detect.py --weights runs/train/exp/weights/best.pt
```

```
--img 640 --conf 0.4 --source ../datasets/traffic/images/test
```

In the case where inference is concerned with the estimated model from the **k**-th experiment, the command to use is:

```
python detect.py --weights runs/train/expk/weights/best.pt
--img 640 --conf 0.4 --source ../datasets/traffic/images/test
```

where the used standard model parameters for detection are:

- **img**: to indicate the input image size. The input images will be automatically resized to the value assigned to this parameter before feeding to the network defined by the estimated model.
- **conf**: to specify the minimum level of confidence to consider for inference.
- **weights**: to specify location of weights associated with the estimated model to use for detection.
- **source**: to specify the path to image or video files on which to make detection.

Each time an inference command similar to the one provided above is executed for an experiment of detecting custom objects on image or video files, results are saved by default to new incrementing directory as follows.

1. All results associated with first experiment of inference are saved by default to the path **yolov5/runs/detect/exp**.
2. All results associated with a subsequent **k**-th experiment of inference are saved by default to the path **yolov5/runs/detect/expk**.

These directories contain input images or video files in which bounding boxes, class labels and confidence levels associated with the target objects of interest are displayed. Interested readers can refer to the website https://github.com/ultralytics/yolov5/issues/36 in order to learn about alternative ways of performing inference based either on a custom trained model or on available pretrained models in the YOLOv5 family. The results associated with inference performed on the test images located at the path specified in the **traffic.yaml** file can be downloaded or visualized in the **detect** folder available at the following location: https://drive.google.com/drive/folders/1RZRnfSMzdnQuM6r-3iCHenl8c0dN3X92?usp=sharing. This location also contains the best trained model named as **best.pt** in the folder called **model**.

Conclusion

The main goal of this report was to build a detector of all objects present in the surrounding of an autonomous vehicle on roads. We achieved this goal by means of a deep learning algorithm called YOLO (You Only Look Once) in the field of computer vision. The basic concepts in object detection as well as the core of the YOLO algorithm are recalled in this study. Our machine learning modeling operations consist of the three following steps. Firstly, we collected a large dataset of images containing 35 classes of traffic scene objects. Secondly, we annotated the collected images in the YOLO format. Thirdly, we fed a model from the YOLO family with the annotated dataset in order to estimate its parameters. The obtained model has pretty good predictive performance and can be used to extract in real-time all information associated with the external driving environment from videos taken by cameras embedded in an autonomous vehicle. Traffic scene elements extracted by such a detector can act as covariates in reliability analysis of automated driving systems consisting to check whether a safety requirement is satisfied in an operational design domain.

Bibliography

[1] R. Atienza. *Advanced Deep Learning with TensorFlow 2 and Keras: Apply DL, GANs, VAEs, deep RL, unsupervised learning, object detection and segmentation, and more, 2nd Edition*. Packt Publishing, 2020 (cited on pages 52, 57, 58).

[2] V. K. Ayyadevara and Y. Reddy. *Modern Computer Vision with PyTorch: Explore deep learning concepts and implement over 50 real-world image applications*. Packt Publishing, 2020 (cited on page 59).

[3] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. URL: https://arxiv.org/abs/2004.10934 (cited on pages 10, 17).

[4] Z. Cai and N. Vasconcelos. *Cascade R-CNN: High Quality Object Detection and Instance Segmentation*. 2019. URL: https://arxiv.org/abs/1906.09756 (cited on page 10).

[5] W. Chen et al. *Deep Learning for Instance Retrieval: A Survey*. 2021. URL: https://arxiv.org/abs/2101.11282 (cited on page 9).

[6] J. Dai et al. *R-FCN: Object Detection via Region-based Fully Convolutional Networks*. 2016. URL: https://arxiv.org/abs/1605.06409 (cited on page 10).

[7] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pages 248–255 (cited on page 24).

[8] M. Everingham et al. "The Pascal Visual Object Classes (VOC) Challenge". In: *International Journal of Computer Vision* 88.2 (June 2010), pages 303–338 (cited on page 24).

[9] M. Everingham et al. "The Pascal Visual Object Classes Challenge: A Retrospective". In: *International Journal of Computer Vision* 111.1 (Jan. 2015), pages 98–136 (cited on page 24).

[10] R. Girshick. *Fast R-CNN*. 2015. URL: https://arxiv.org/abs/1504.08083 (cited on page 10).

[11] R. Girshick et al. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2013. URL: https://arxiv.org/abs/1311.2524 (cited on page 10).

[12] K. He et al. *Mask R-CNN*. 2017. URL: https://arxiv.org/abs/1703.06870 (cited on page 24).

[13] A. G. Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. URL: https://arxiv.org/abs/1704.04861 (cited on page 24).

[14] J. Janai et al. *Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art*. 2017. URL: https://arxiv.org/abs/1704.05519 (cited on page 9).

[15] J. Kaur and W. Singh. "Tools, techniques, datasets and application areas for object detection in an image: a review". In: *Multimedia Tools and Applications* (2022). DOI: https://doi.org/10.1007/s11042-022-13153-y (cited on page 9).

[16] V. Lakshmanan, M. Görner, and R. Gillard. *Practical Machine Learning for Computer Vision*. O'Reilly Media, 2021 (cited on page 65).

[17] Z. Li and F. Zhou. *FSSD: Feature Fusion Single Shot Multibox Detector*. 2017. URL: https://arxiv.org/abs/1712.00960 (cited on page 10).

[18] T-Y Lin et al. *Microsoft COCO: Common Objects in Context*. 2014. URL: https://arxiv.org/abs/1405.0312 (cited on page 24).

[19] T.-Y. Lin et al. *Focal Loss for Dense Object Detection*. 2017. URL: https://arxiv.org/abs/1708.02002 (cited on page 10).

[20] S. Liu, D. Huang, and Y. Wang. *Receptive Field Block Net for Accurate and Fast Object Detection*. 2017. URL: https://arxiv.org/abs/1711.07767 (cited on page 10).

[21] W. Liu et al. "SSD: Single Shot MultiBox Detector". In: *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, pages 21–37. URL: https://doi.org/10.1007%2F978-3-319-46448-0_2 (cited on page 10).

[22] R. Padilla et al. "A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit". In: *Electronics* 10.3 (2021). DOI: 10.3390/electronics10030279 (cited on page 15).

[23] A. R. Pathak, M. Pandey, and S. Rautaray. "Application of Deep Learning for Object Detection". In: *Procedia Computer Science* 132 (2018). International Conference on Computational Intelligence and Data Science, pages 1706–1717. DOI: https://doi.org/10.1016/j.procs.2018.05.144 (cited on page 9).

[24] J. Redmon and A. Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. URL: https://arxiv.org/abs/1612.08242 (cited on pages 10, 17, 68).

[25] J. Redmon and A. Farhadi. *YOLOv3: An Incremental Improvement*. 2018. URL: https://arxiv.org/abs/1804.02767 (cited on pages 10, 17).

[26] J. Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2015. URL: https://arxiv.org/abs/1506.02640 (cited on pages 10, 17, 69).

[27] S. Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2015. URL: https://arxiv.org/abs/1506.01497 (cited on page 10).

[28] R. Szeliski. *Computer Vision - Algorithms and Applications, Second Edition*. Texts in Computer Science. Springer, 2022 (cited on page 64).

[29] A. Vahab, M. S. Naik, and P. G. Raikar. "Applications of Object Detection System". In: *International Research Journal of Engineering and Technology* 6 (2019). URL: https://www.irjet.net/archives/V6/i4/IRJET-V6I4920.pdf (cited on page 9).

[30] V. Verdhan. *Computer Vision Using Deep Learning: Neural Network Architectures with Python and Keras*. Apress, 2021 (cited on page 56).

[31] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. 2022. URL: https://arxiv.org/abs/2207.02696 (cited on pages 10, 17).

A. Supplementary results for Chapter 1

A.1 Illustration of bounding boxes

Bounding boxes (part 1/4)

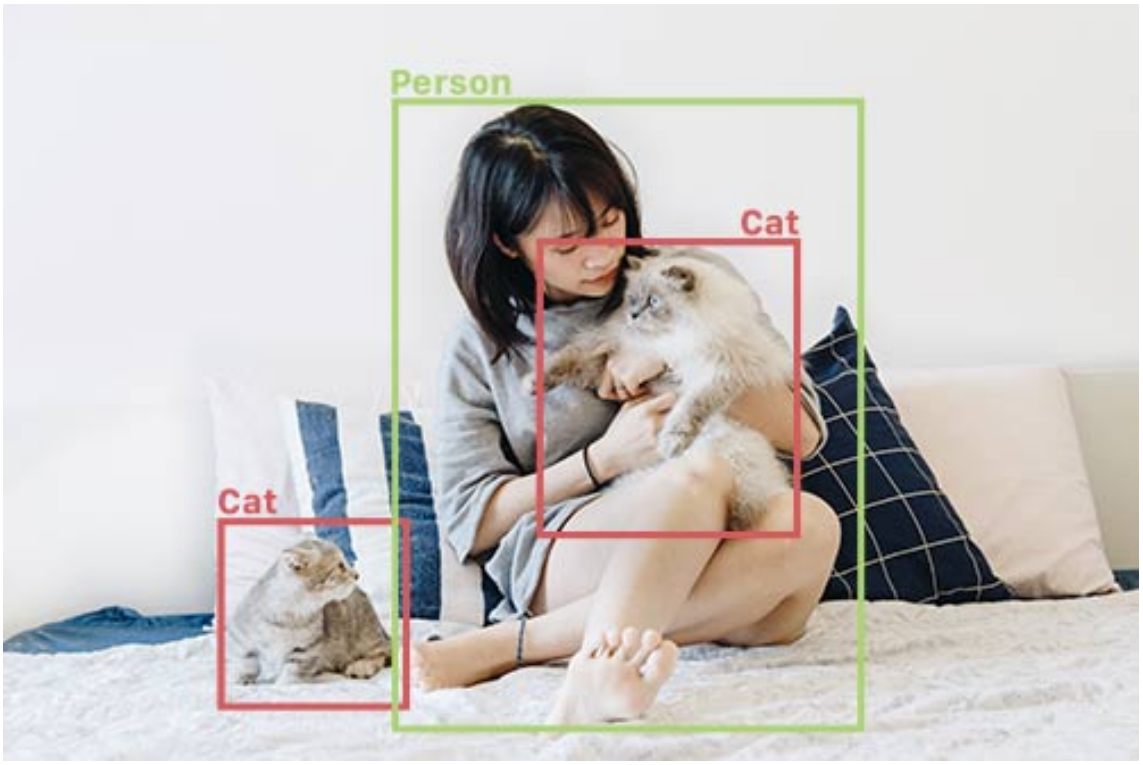


Figure A.1: Illustration of an object detector in action (source: https://www.fritz.ai/object-detection/).

Bounding boxes (part 2/4)

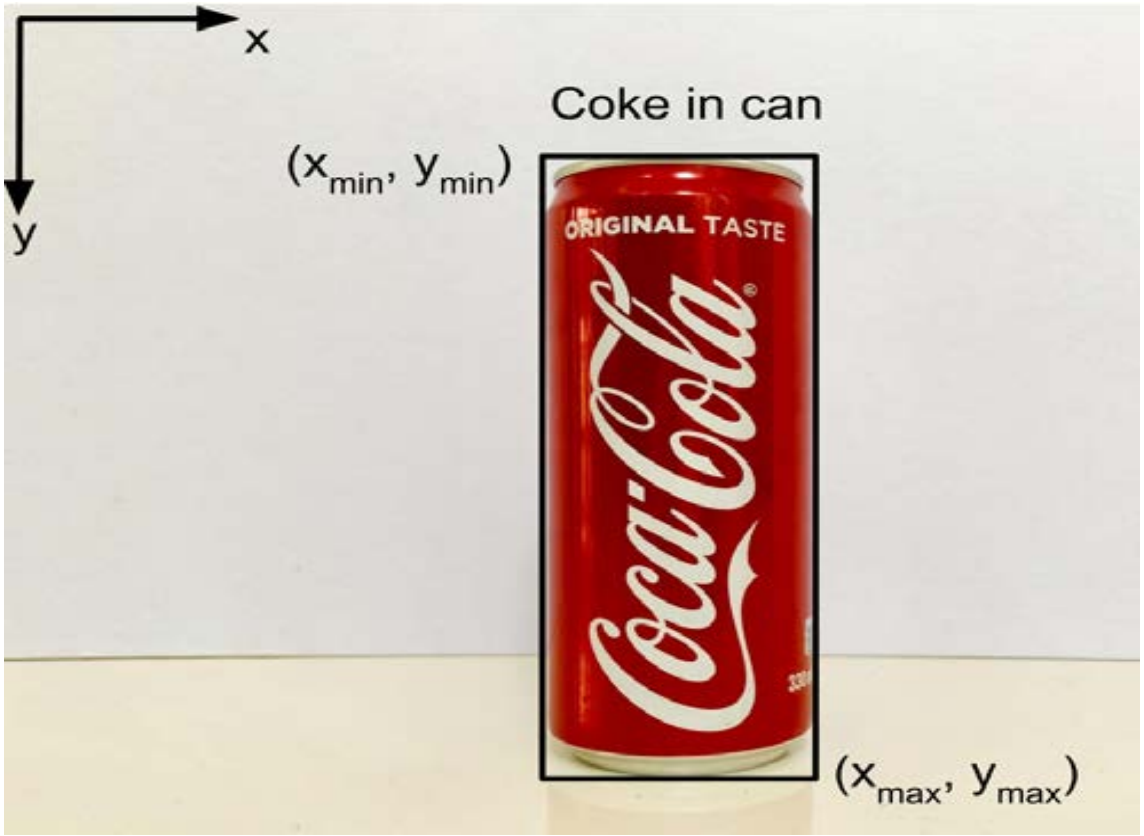


Figure A.2: Illustration of the bounding box coordinates of an object from the **Coke in can** class. (source: Figure 11.1.1 from the book [1]).

Bounding boxes (part 3/4)

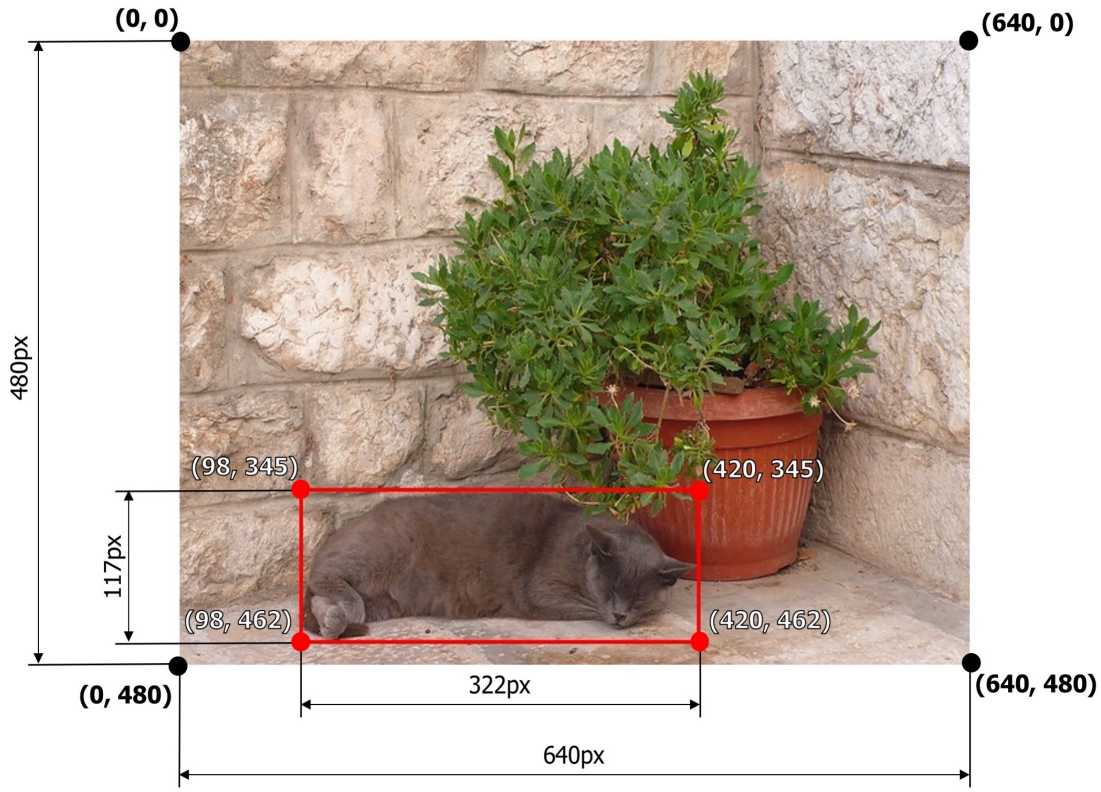


Figure A.3: Illustration of the bounding box coordinates of an object from the **cat** class. (source: https://albumentations.ai/docs/getting_started/bounding_boxes_augmentation/).

Bounding boxes (part 4/4)

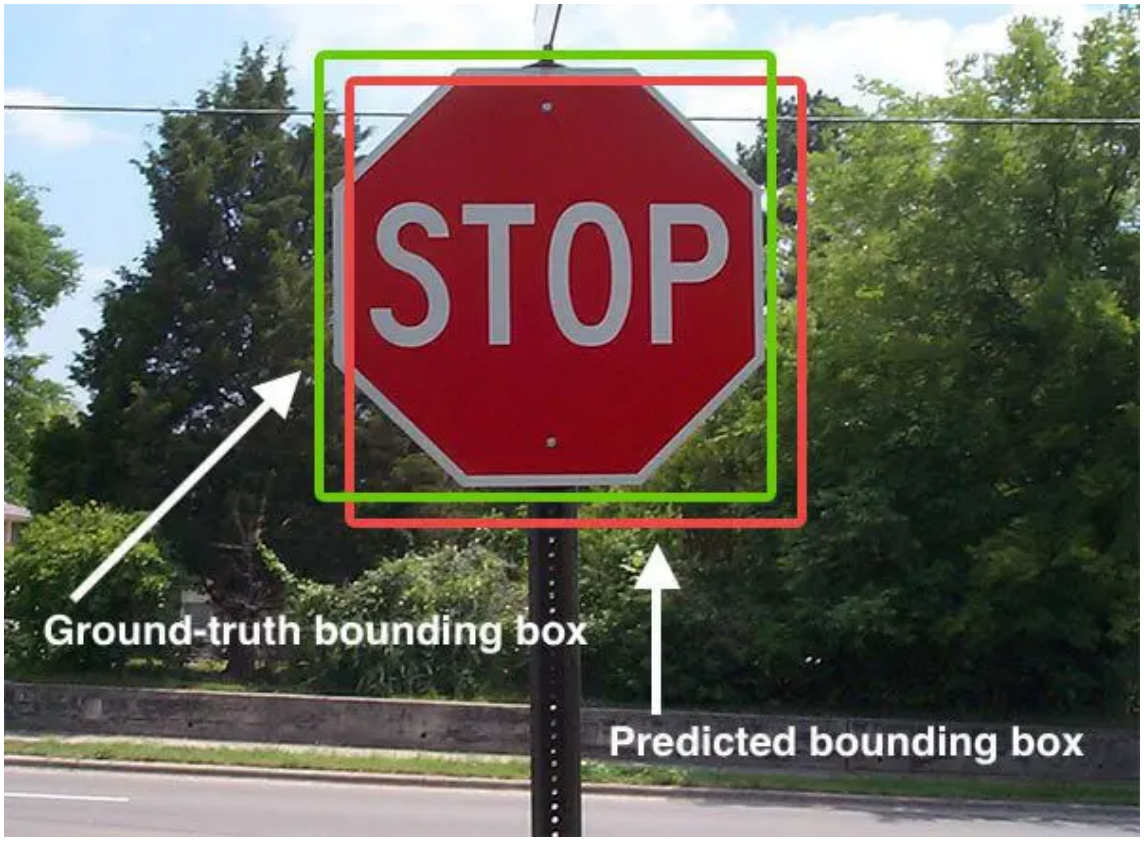


Figure A.4: Illustration of the predicted and the ground-truth bounding boxes associated with a **stop sign** in an image (source: https://en.wikipedia.org/wiki/Jaccard_index).

A.2 Illustration of intersection over union

Intersection over Union (part 1/2)

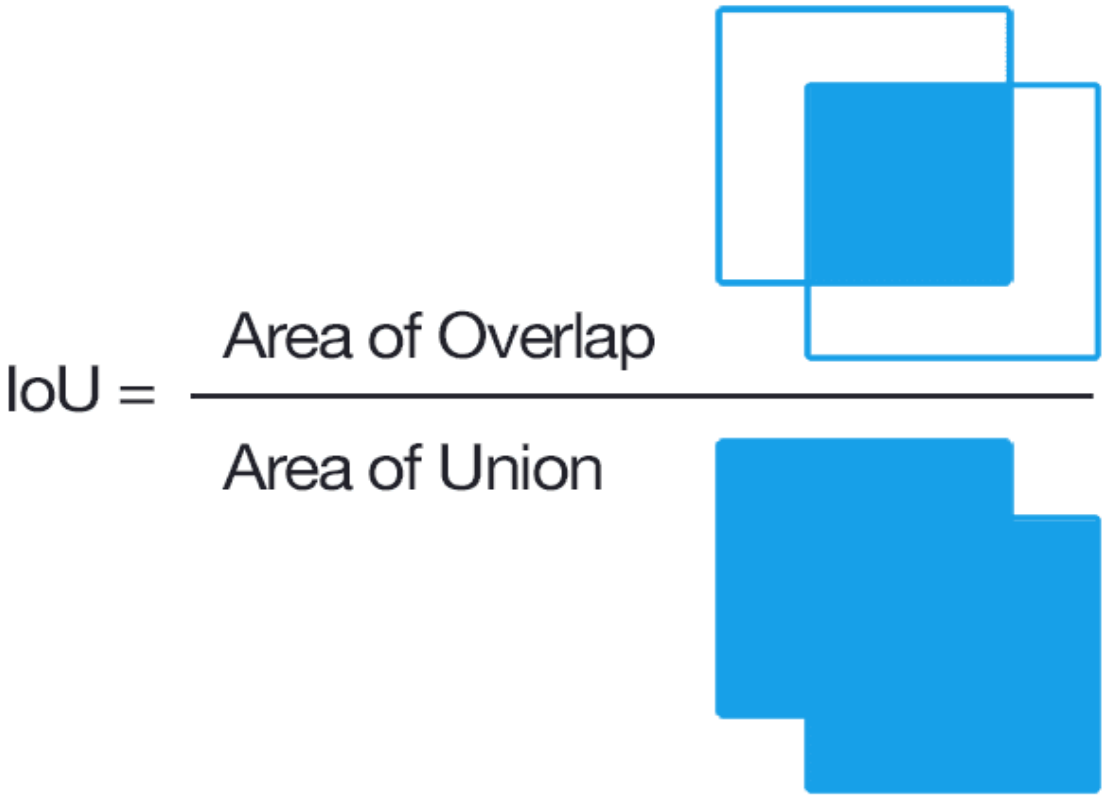


Figure A.5: Illustration of the Intersection over Union as ration of the two areas in blue. One can assume that the left and the right bounding boxes are respectively the ground-truth and the predicted locations of a target object (source: https://en.wikipedia.org/wiki/Jaccard_index).

Intersection over Union (part 2/2)

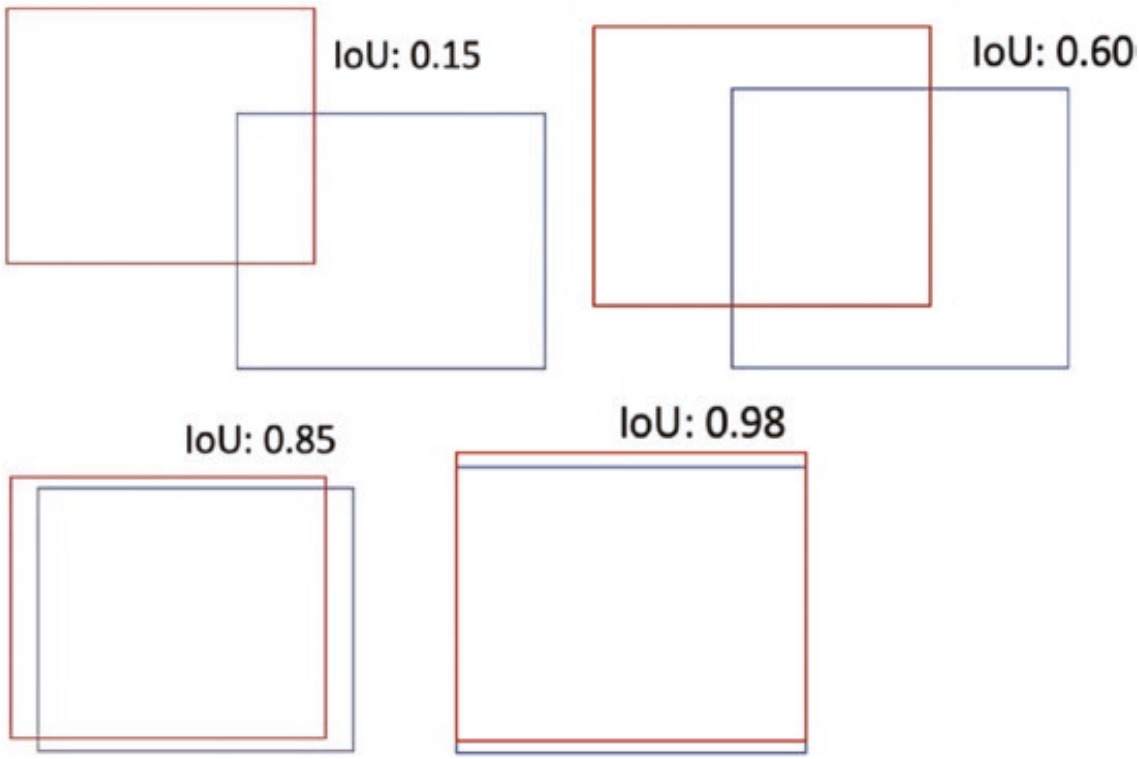


Figure A.6: Illustration of IoU in four different cases. The numbers are the IoU between two bounding boxes indicating how well they overlap with each other. One can assume that the gray and the red bounding boxes are respectively the ground-truth and the predicted locations of a target object (source: Figure 5-5 from the book [30]).

A.3 Illustration of anchor box

Anchor box (part 1/3)

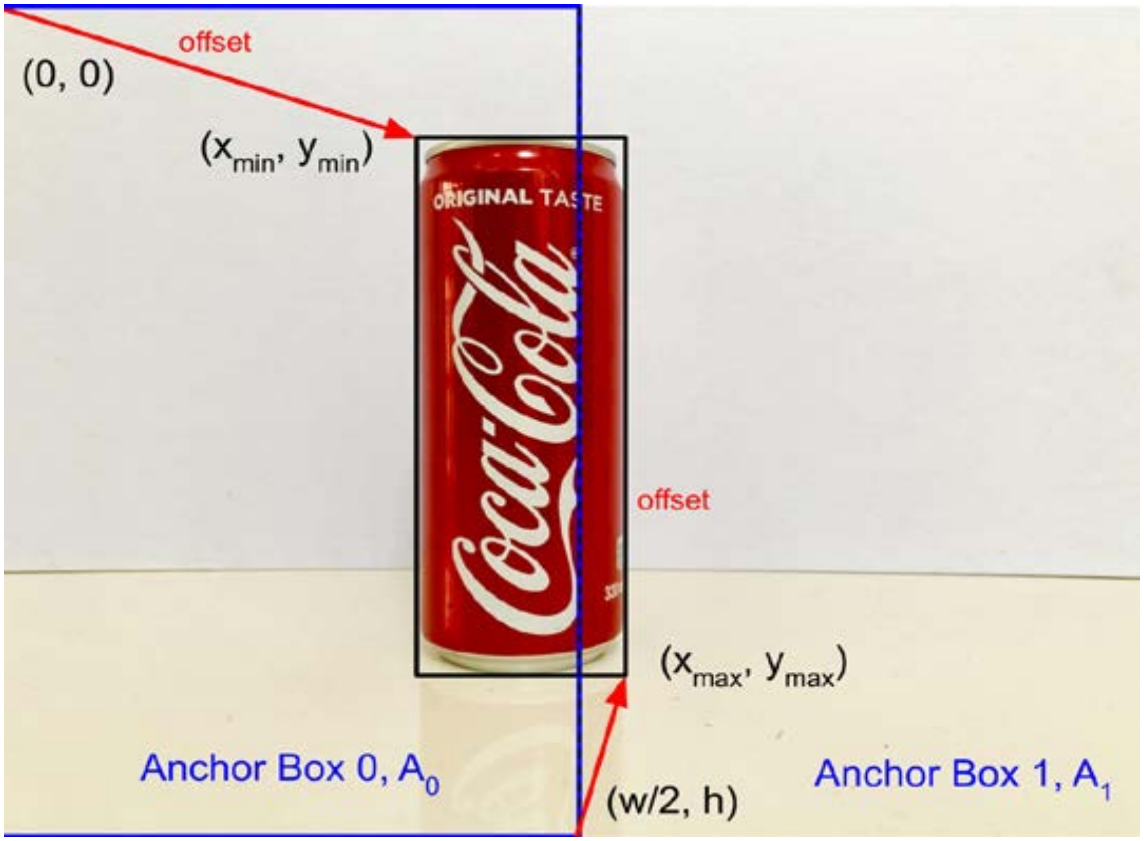


Figure A.7: Examples of anchor boxes and offsets. In the first anchor box, the coordinates of the offsets are $(x_{\min}, y_{\min})$ and $(x_{\max} - w/2, y_{\max} - h)$, where w and h are the image width and height, respectively. (source: Figure 11.2.1 from the book [1]).

Anchor box (part 2/3)

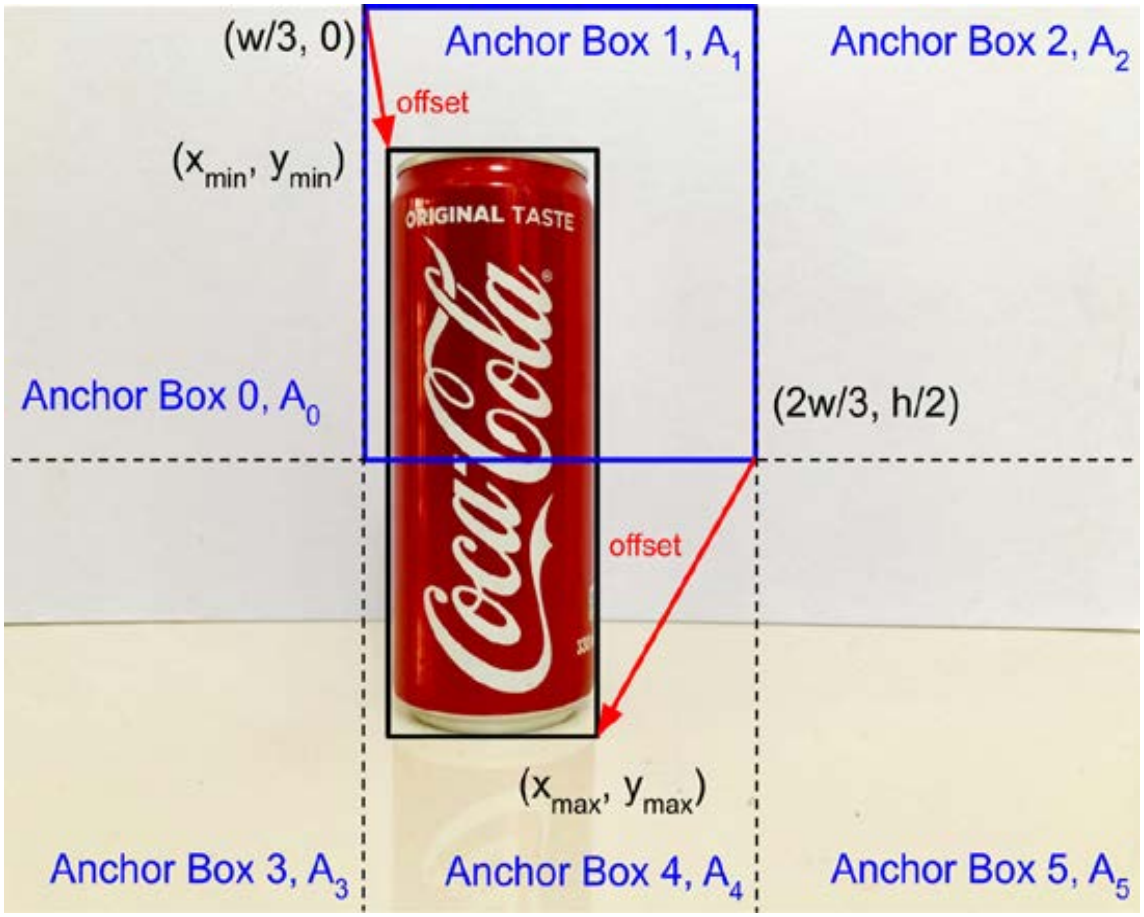


Figure A.8: Examples of anchor boxes and offsets. In the second anchor box, the coordinates of the offsets are $(x_{\min} - w/3, y_{\min})$ and $(x_{\max} - 2w/3, y_{\max} - h/2)$, where w and h are the image width and height, respectively. (source: Figure 11.2.2 from the book [1]).

Anchor box (part 3/3)

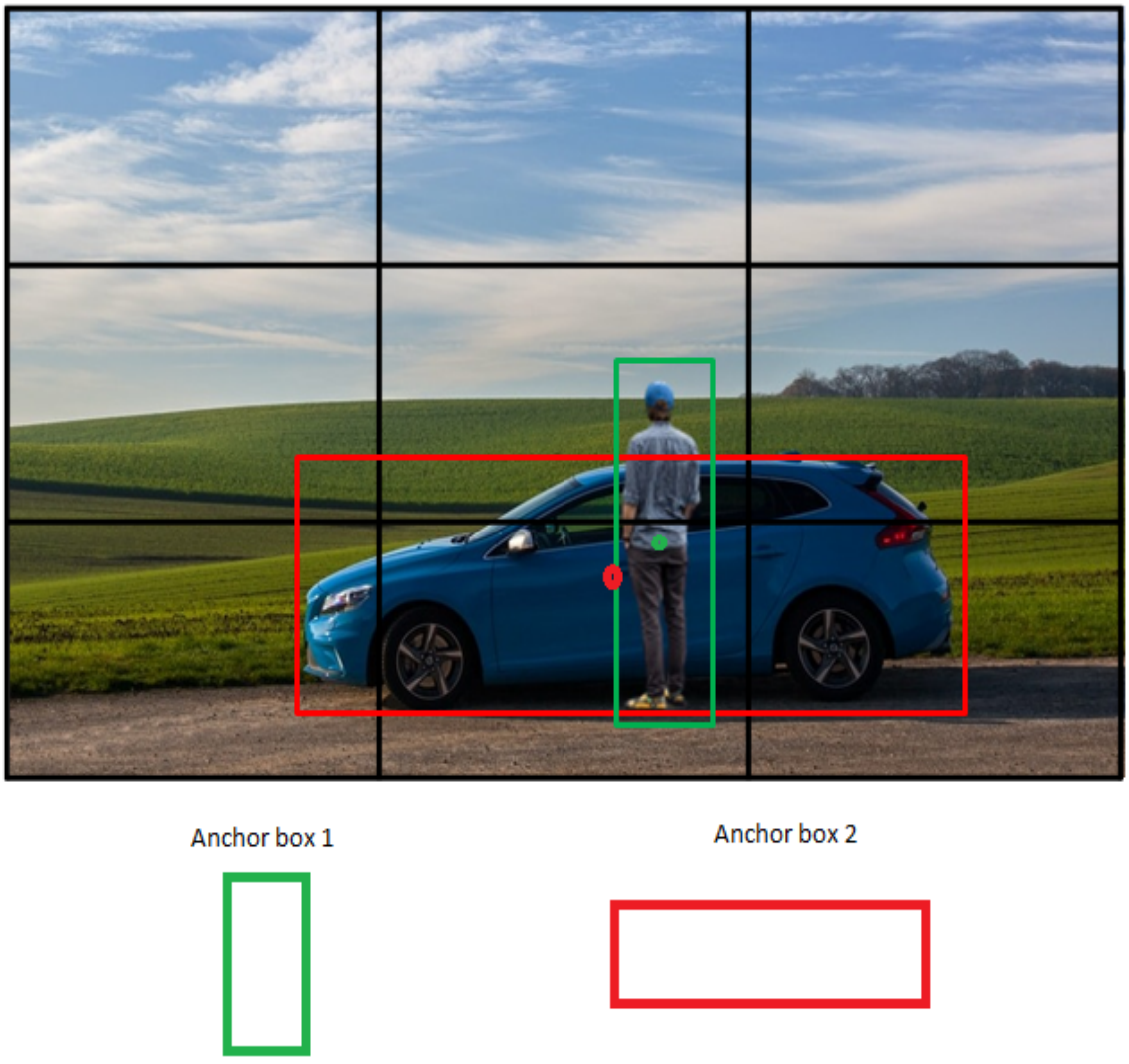


Figure A.9: Example of an anchor box suitable for an object of a **person** class and an anchor box suitable for an object of a **car** class (source: Figure of the page 355 from the book [2]).

A.4 Illustration of non-maximum suppression

Non-Maximum Suppression (part 1/2)

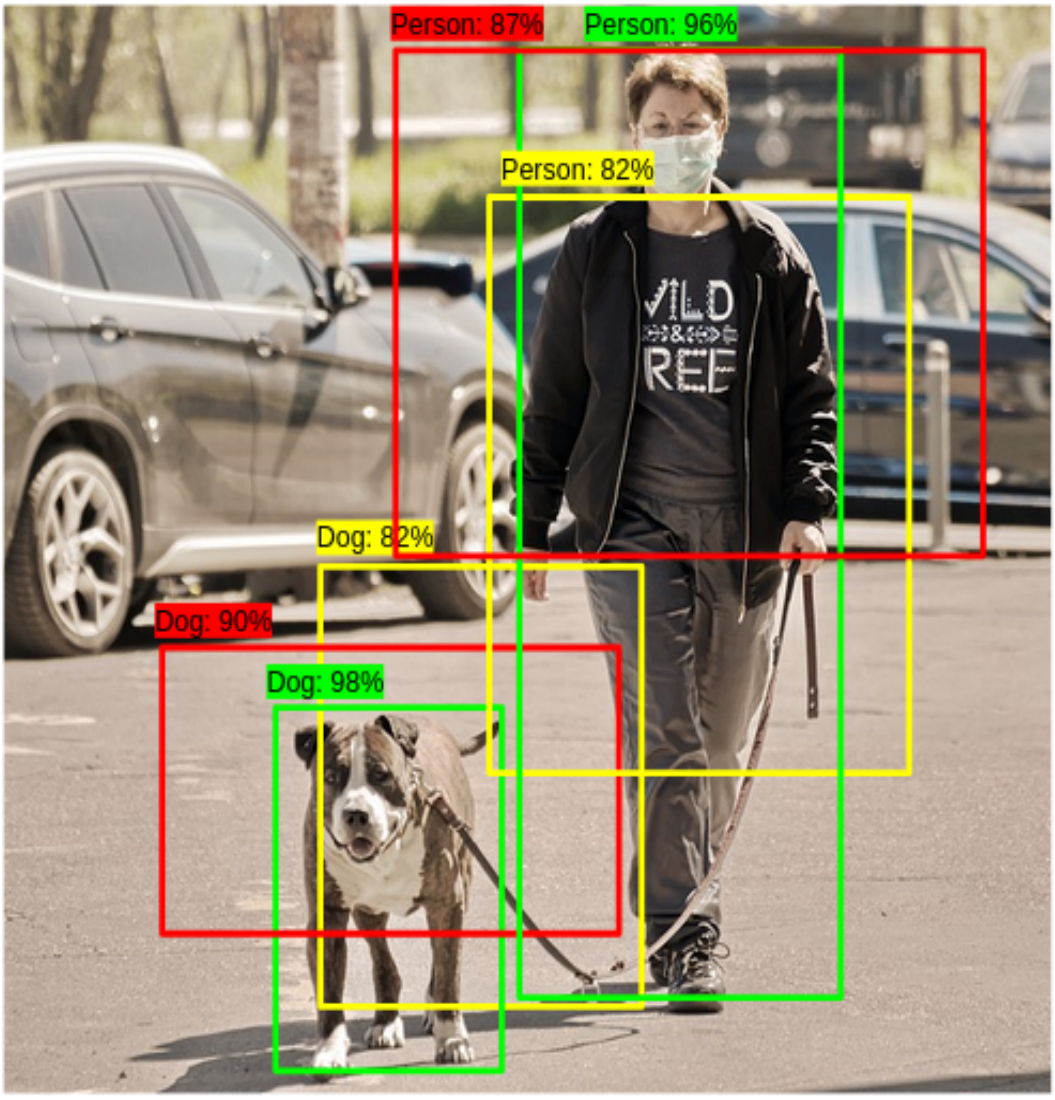


Figure A.10: Example of an image with many overlapping bounding boxes associated with an object from a **Person** class and an object from a **Dog** class. The confidence score (in percentage) of bounding boxes are also displayed. (source: https://www.analyticsvidhya.com/blog/2020/08/selecting-the-right-bounding-box-using-non-max-suppression-with-implementation/).

Non-Maximum Suppression (part 2/2)

A.5 Illustration of confusion matrix

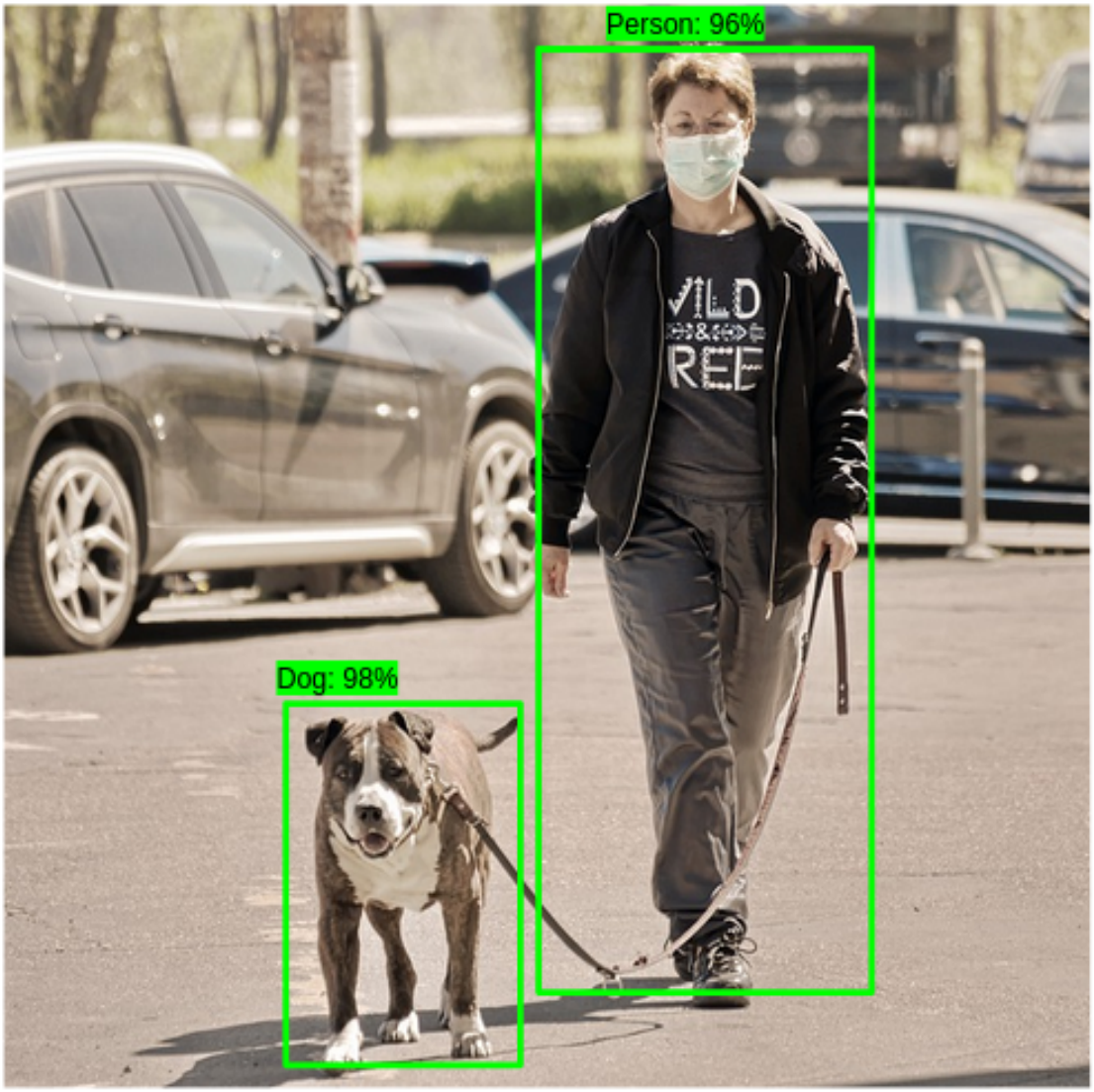


Figure A.11: Bounding boxes selected by the Non-Maximum Suppression Procedure 1 applied to those provided in the Figure A.10. (source: https://www.analyticsvidhya.com/blog/2020/08/selecting-the-right-bounding-box-using-non-max-suppression-with-implementation/).

A.6 Illustration of detection quality

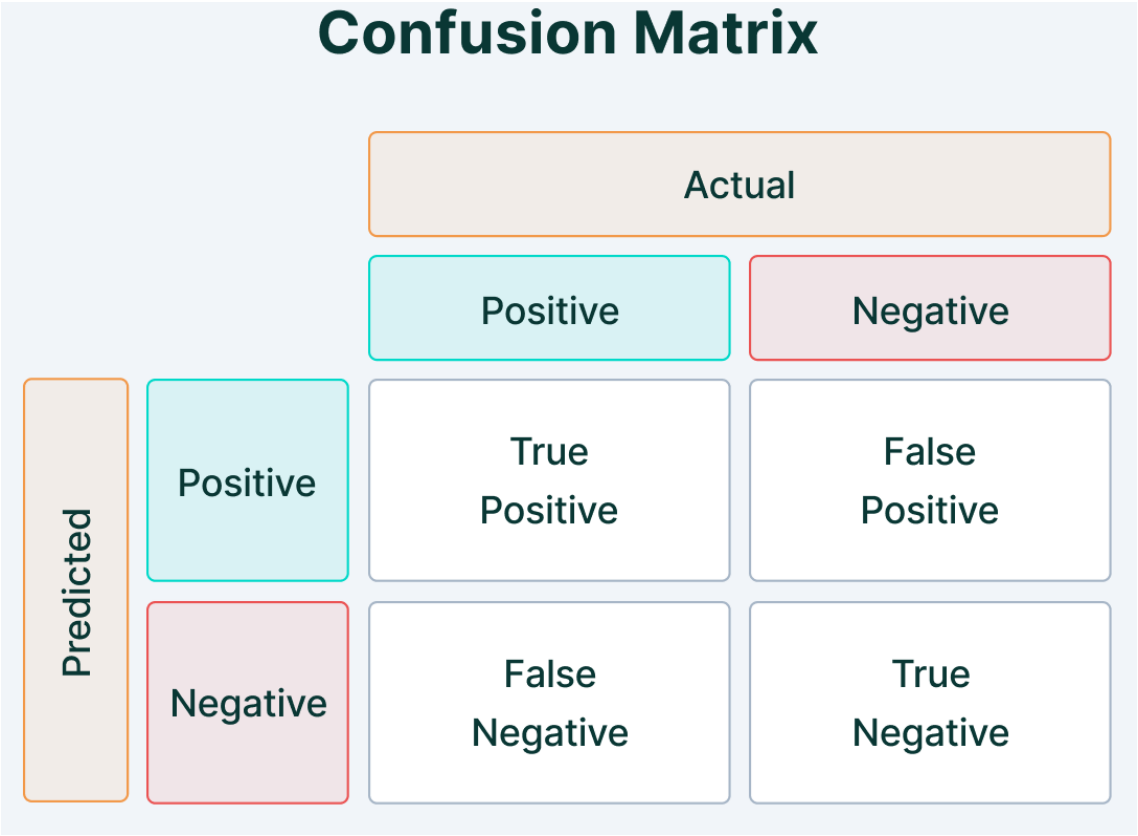


Figure A.12: A 2×2 confusion matrix to evaluate and visualize the performance of any binary classification models into positive and negative classes. (source: https://www.v7labs.com/blog/mean-average-precision).

A.7 Illustration of Average Precision

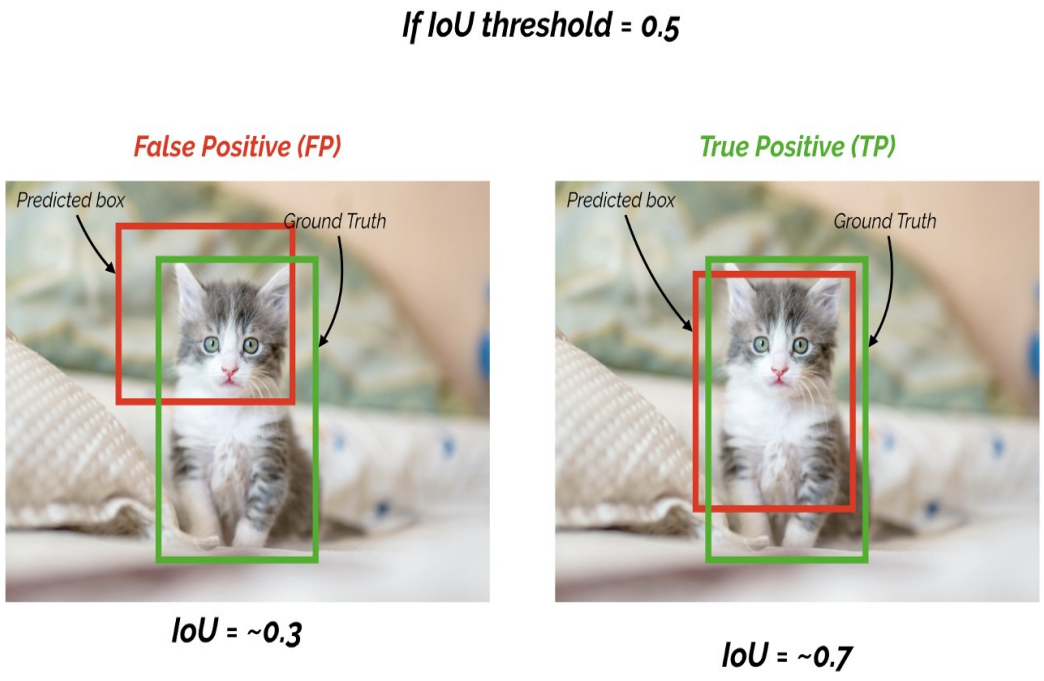


Figure A.13: Quality of a detection when the IoU threshold is equal to 0.5. The detection of the cat on the left image is considered as false positive since the IoU between the predicted and the ground-truth bounding boxes is equal to 0.3 (< 0.5). In contrast, the detection of the cat on the right image is considered as true positive since the IoU between the predicted and the ground-truth bounding boxes is equal to 0.7 (> 0.5.) (source: https://www.v7labs.com/blog/mean-average-precision).

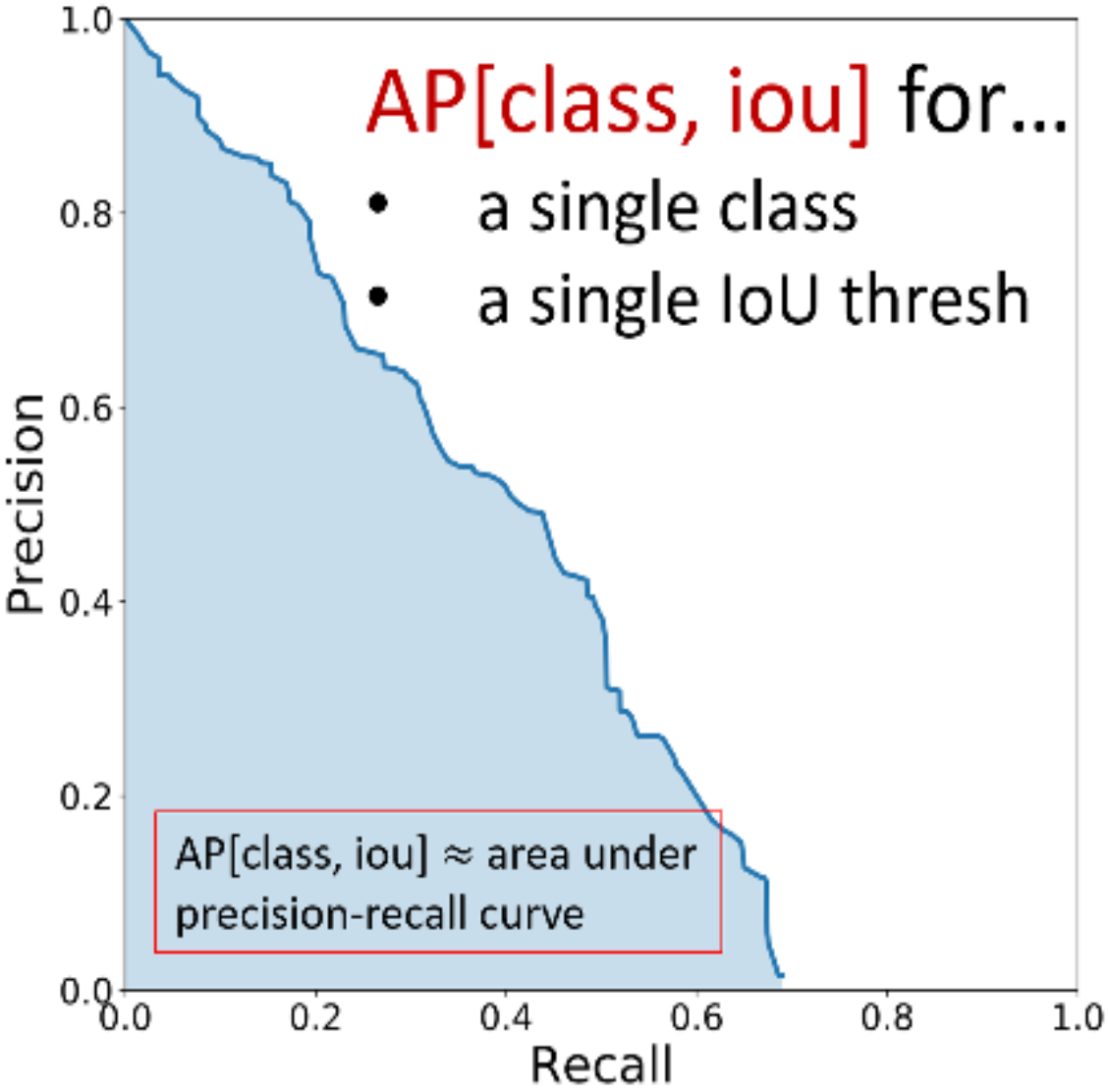


Figure A.14: A precision-recall curve of an object detector for a single class and IoU threshold in which the average precision (AP) is indicated as the area under that curve (source: Figure 6.27 (a) from the book [28]).

B. Supplementary results for Chapter 2

B.1 Illustration of positive grid cell

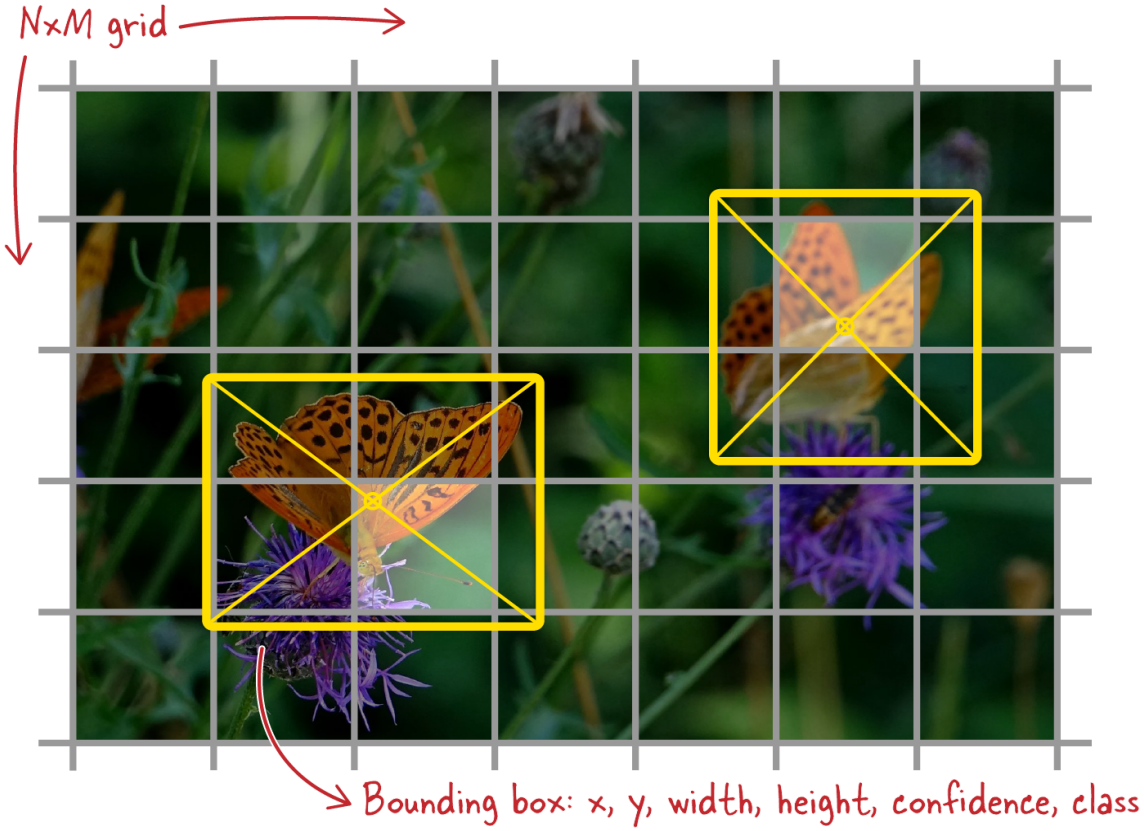


Figure B.1: Image divided into 7×5 grid cells. Each grid cell predicts bounding boxes of objects whose centers are somewhere in that cell. The two highlighted cells are those containing the centers of the ground-truth bounding boxes in yellow associated with objects from the **butterfly** class. (Source: Figure 4-4 from the book [16]).

B.2 Illustration of anchor boxes design

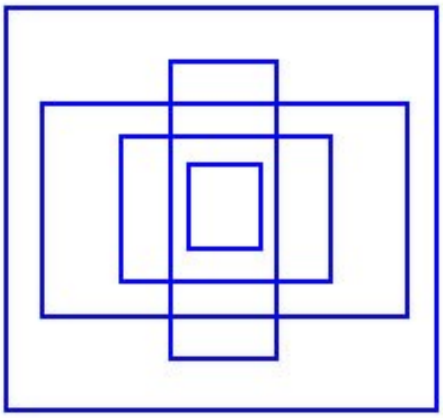


Figure B.2: Example of five anchor boxes of different dimensions which can result from a k-means clustering of all ground-truth bounding boxes. (Source: https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088).

B.3 Illustration of responsible anchor box

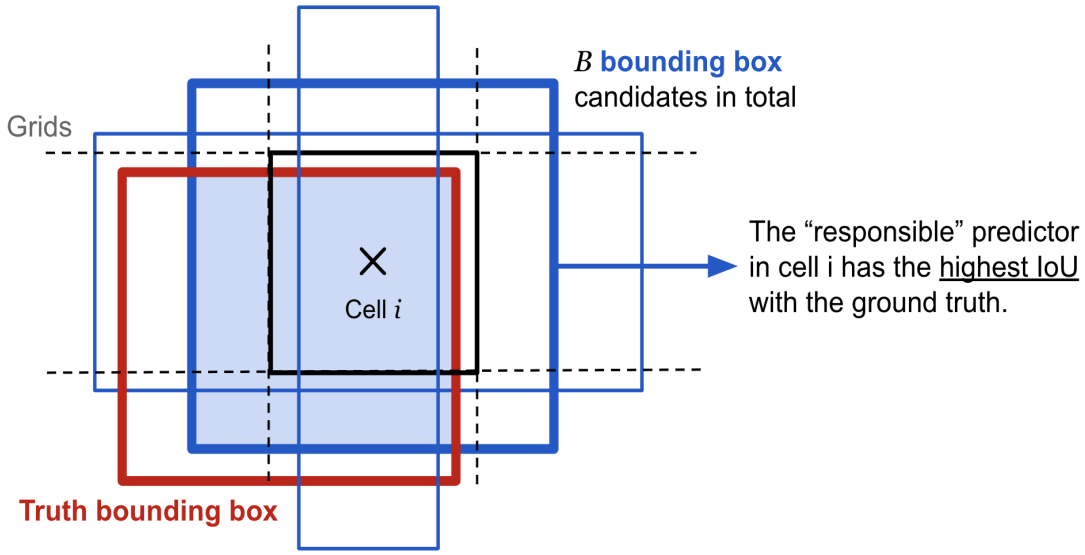


Figure B.3: Example of an anchor box selected among the $B = 3$ candidate anchor boxes in blue as the one having the highest IoU with the ground truth bounding box in red. Note that the cell i in black is responsible for the prediction of the ground-truth bounding box and that all candidate anchor boxes have their centers in the cell i. Therefore, the selected anchor box is said to be *responsible* for the prediction of the ground-truth bounding box whereas the non selected anchor boxes are said to be not *responsible* for the prediction of an object in the cell i. (Source: https://lilianweng.github.io/posts/2018-12-27-object-recognition-part-4/).

B.4 Illustration of offsets in YOLO modeling

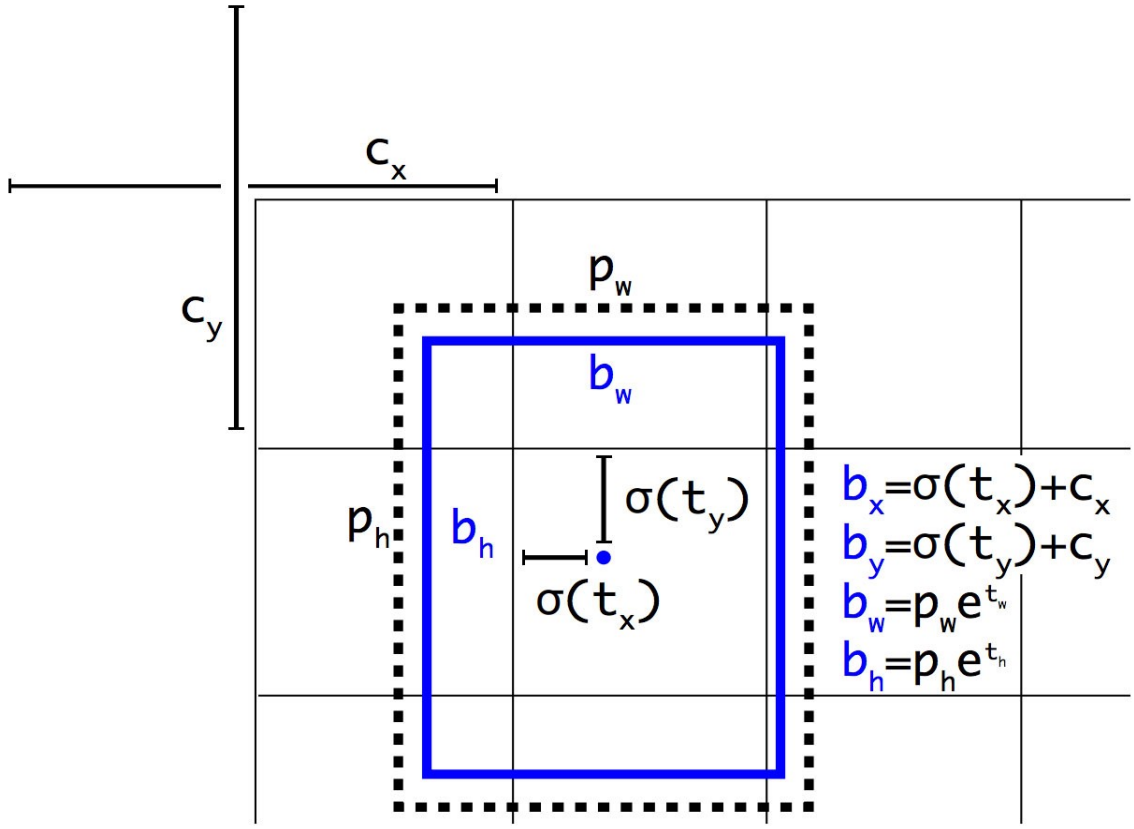


Figure B.4: YOLO model representation of the offsets between the anchor box (dotted rectangle) and the predicted bounding box (blue rectangle). Here, $\sigma(\cdot)$ is the sigmoid function. (Source: Figure 3 from the paper [24]).

B.5 Illustration of YOLO detection pipeline

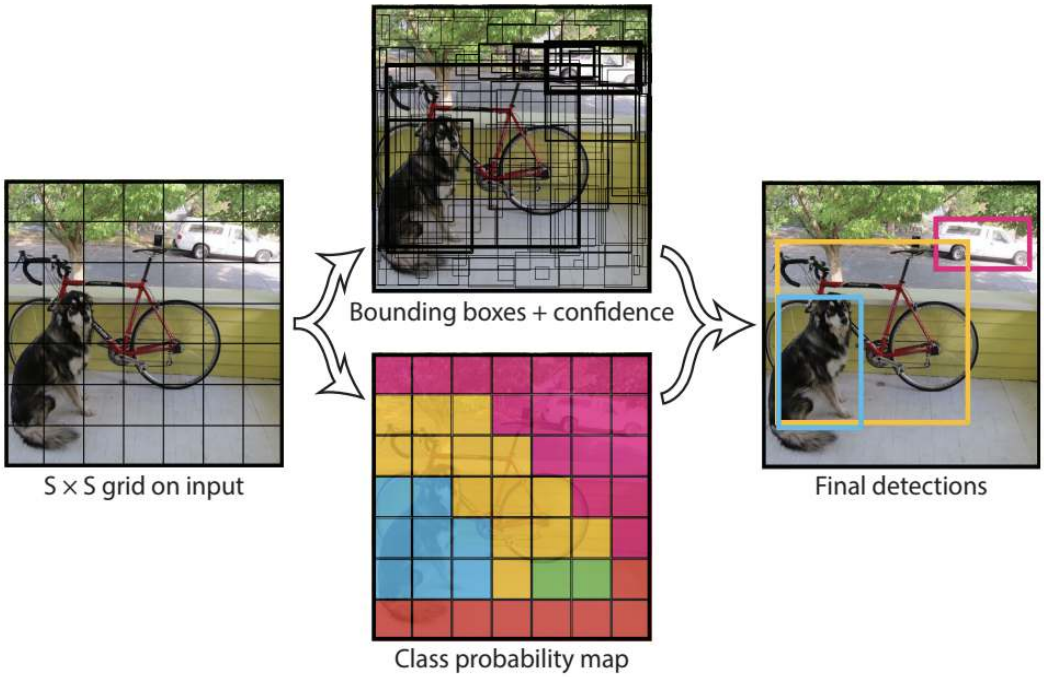


Figure B.5: A simplified illustration of the YOLO object detector pipeline in an image. (Source: Figure 2 from the paper [26]).

C. Supplementary results for Chapter 3

C.1 Class of car

Figure C.1: Sample objects from the class of **car**.

C.2 Class of car group

Figure C.2: Sample objects from the class of **car group**.

C.3 Class of bus

Figure C.3: Sample objects from the class of **bus**.

C.4 Class of truck

Figure C.4: Sample objects from the class of **truck**.

C.5 Class of train

Class of train (part 1/2)

Figure C.5: Sample objects from the class of **train** (part 1/2).

Class of train (part 2/2)

Figure C.6: Sample objects from the class of **train** (part 2/2).

C.6 Class of airplane

Figure C.7: Sample objects from the class of **airplane**.

C.7 Class of helicopter

Figure C.8: Sample objects from the class of **helicopter**.

C.8 Class of boat

Figure C.9: Sample objects from the class of **boat**.

C.9 Class of scooter

Figure C.10: Sample objects from the class of **scooter**.

C.10 Class of bicycle

Figure C.11: Sample objects from the class of **bicycle**.

C.11 Class of motorcycle

Figure C.12: Sample objects from the class of **motorcycle**.

C.12 Class of tunnel entrance

Figure C.13: Sample objects from the class of **tunnel entrance**.

C.13 Class of tunnel

Figure C.14: Sample objects from the class of **tunnel**.

C.14 Class of fence

Figure C.15: Sample objects from the class of **fence**.

C.15 Class of street light

Figure C.16: Sample objects from the class of **street light**.

C.16 Class of traffic light

Figure C.17: Sample objects from the class of **traffic light**.

C.17 Class of stop sign

Figure C.18: Sample objects from the class of **stop sign**.

C.18 Class of traffic sign

Figure C.19: Sample objects from the class of **traffic sign**.

C.19 Class of fire hydrant

Figure C.20: Sample objects from the class of **fire hydrant**.

C.20 Class of parking meter

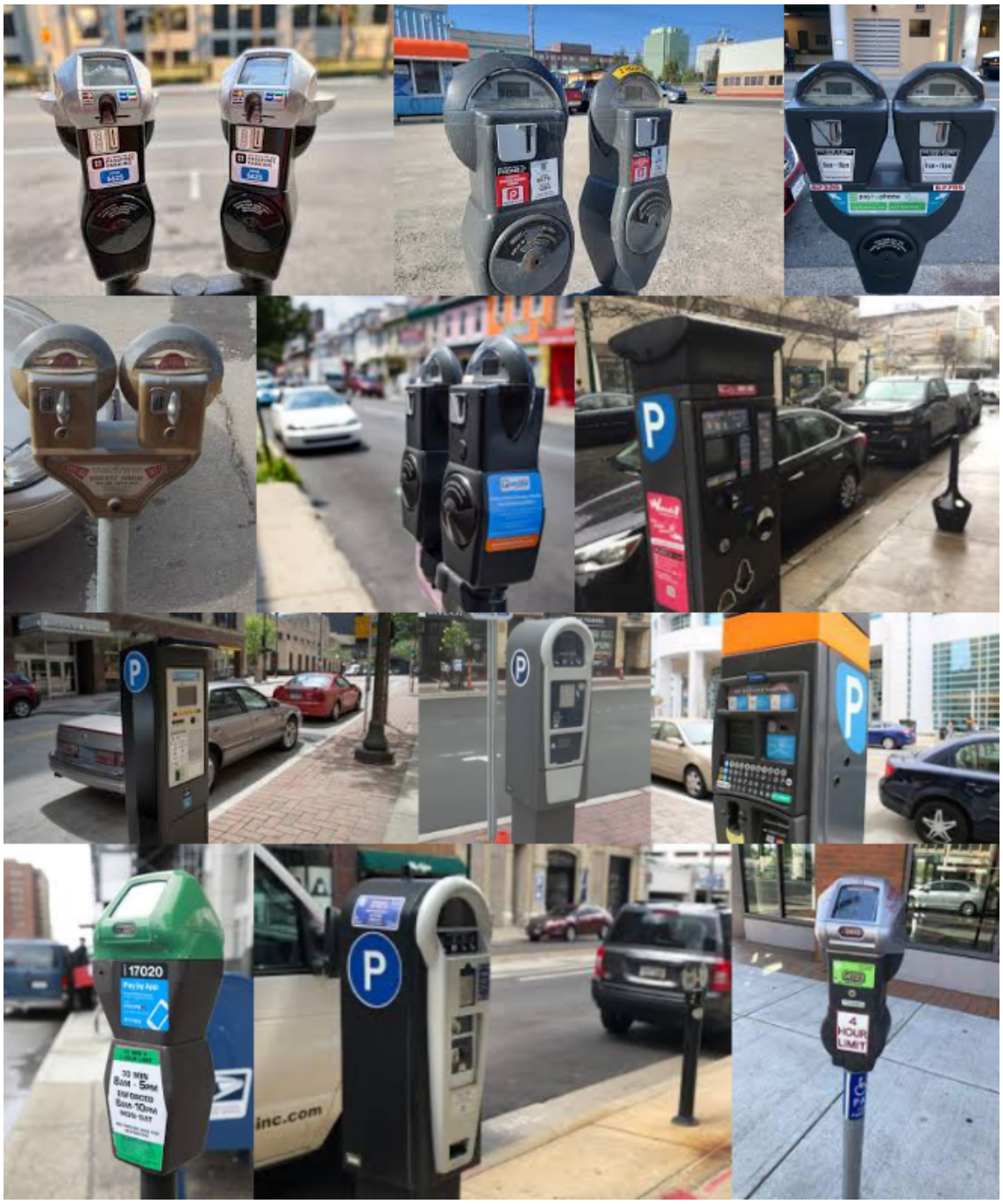


Figure C.21: Sample objects from the class of **parking meter**.

C.21 Class of electric pole

Figure C.22: Sample objects from the class of **electric pole**.

C.22 Class of tree

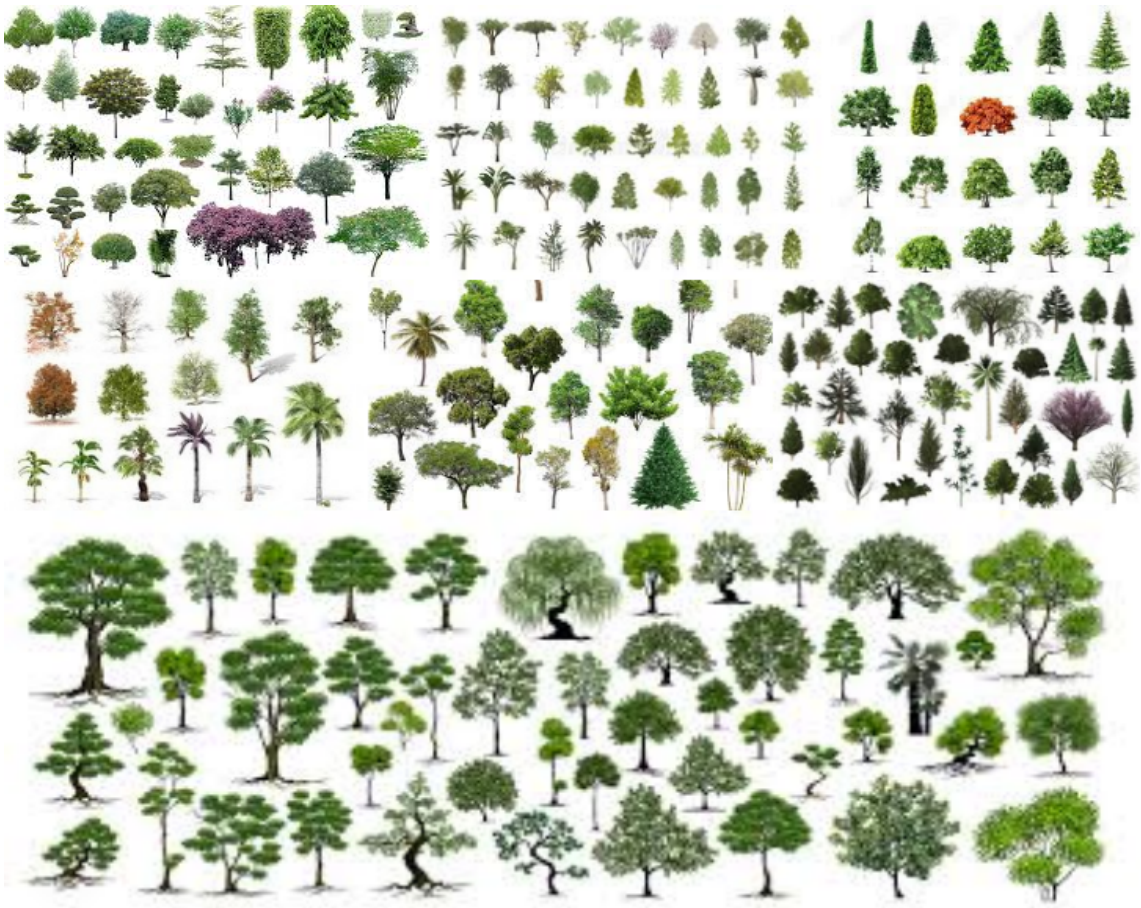


Figure C.23: Sample objects from the class of **tree**.

C.23 Class of tree group

Figure C.24: Sample objects from the class of **tree group**.

C.24 Class of car bench

Figure C.25: Sample objects from the class of **bench**.

C.25 Class of bench group

Figure C.26: Sample objects from the class of **bench group**.

C.26 Class of bird

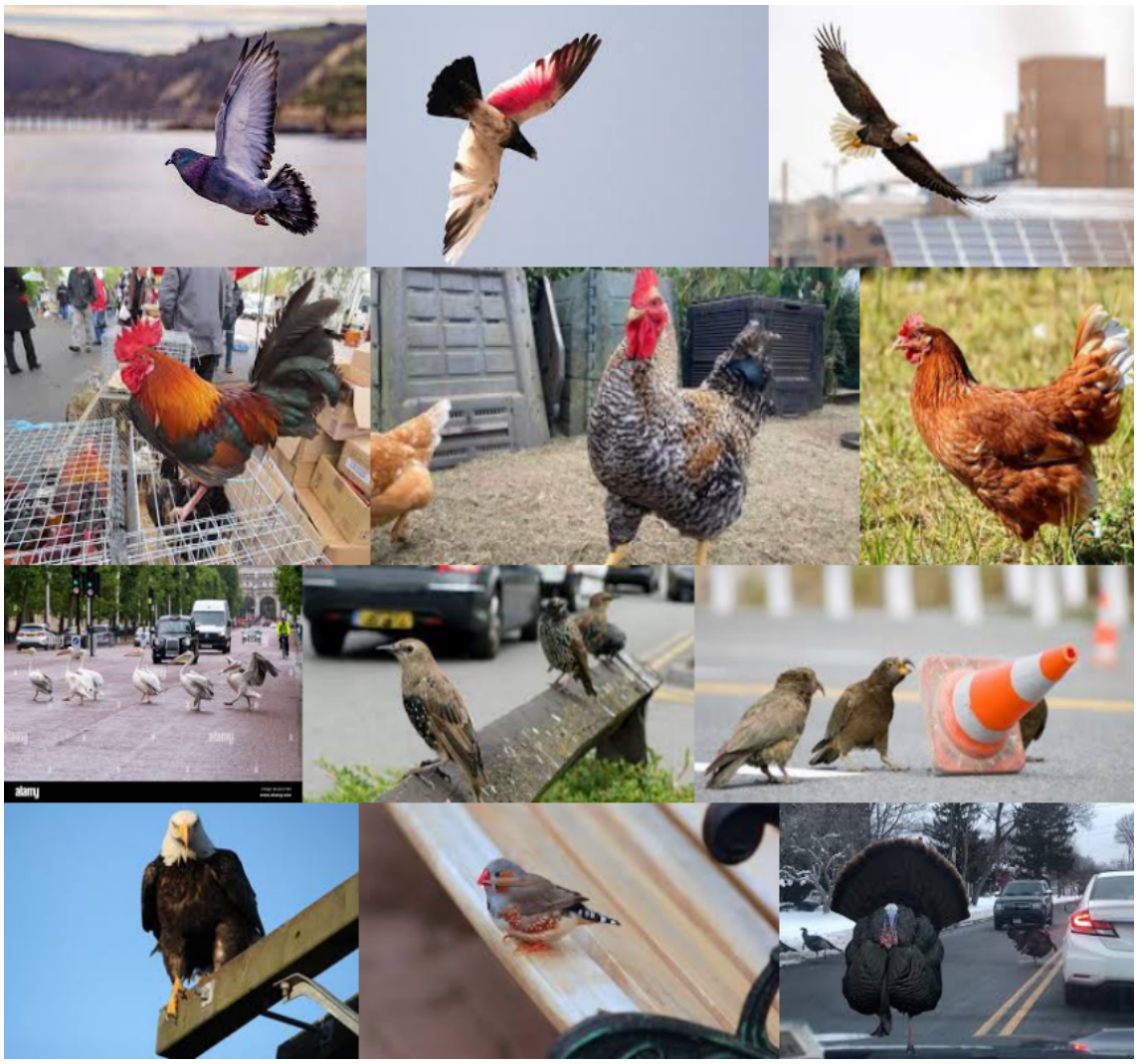


Figure C.27: Sample objects from the class of **bird**.

C.27 Class of bird group

Figure C.28: Sample objects from the class of **bird group**.

C.28 Class of dog

Figure C.29: Sample objects from the class of **dog**.

C.29 Class of person

Figure C.30: Sample objects from the class of **person**.

C.30 Class of person group

Figure C.31: Sample objects from the class of **person group**.

C.31 Class of house

Figure C.32: Sample objects from the class of **house**.

C.32 Class of house group

Figure C.33: Sample objects from the class of **house group**.

C.33 Class of tenement

Figure C.34: Sample objects from the class of **tenement**.

C.34 Class of special building

Class of special building (part 1/3)

Figure C.35: Sample objects from the class of **special building** (part 1/3).

Class of special building (part 2/3)

Figure C.36: Sample objects from the class of **special building** (part 2/3).

Class of special building (part 3/3)

Figure C.37: Sample objects from the class of **special building** (part 3/3).

C.35 Class of gas station

Figure C.38: Sample objects from the class of **gas station**.

C.36 Few annotated images

Figure C.39: File having the name **images1.jpg** used to illustrated the annotation formats in section 3.4.2. A bounding box is drawn around each object of interest.

Figure C.40: File having the name **images2.jpg** used to illustrated the annotation formats in section 3.4.2. A bounding box is drawn around each object of interest.

Figure C.41: File having the name **images3.jpg** used to illustrated the annotation formats in section 3.4.2. A bounding box is drawn around each object of interest.

Figure C.42: File having the name **images4.jpg** used to illustrated the annotation formats in section 3.4.2. A bounding box is drawn around each object of interest.

Figure C.43: File having the name **images5.jpg** used to illustrated the annotation formats in section 3.4.2. A bounding box is drawn around each object of interest.

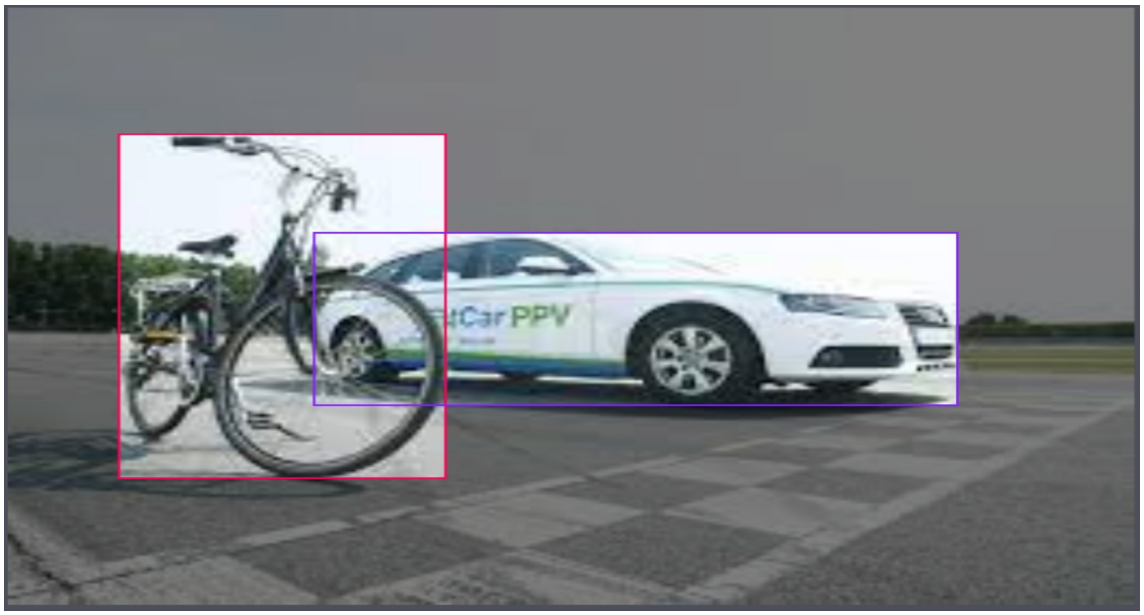


Figure C.44: File having the name **images6.jpg** used to illustrated the annotation formats in section 3.4.2. A bounding box is drawn around each object of interest.

D. Supplementary results for Chapter 4

D.1 Pretrained YOLOv5 models

Model	size (pixels)	mAP^{val} 50-95	mAP^{val} 50	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)
YOLOv5n	640	28.0	45.7	**45**	**6.3**	**0.6**	**1.9**
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7
YOLOv5n6	1280	36.0	54.4	153	8.1	2.1	3.2
YOLOv5s6	1280	44.8	63.7	385	8.2	3.6	12.6
YOLOv5m6	1280	51.3	69.3	887	11.1	6.8	35.7
YOLOv5l6	1280	53.7	71.3	1784	15.8	10.5	76.8
YOLOv5x6 + TTA	1280 1536	55.0 **55.8**	72.7 **72.7**	3136 -	26.2 -	19.4 -	140.7 -

Figure D.1: Performances of models in YOLOv5 family with respect to validation data associated with the MS COCO data set (source: https://github.com/ultralytics/yolov5).

D.2 Description of all input images

D.2.1 Numerical summaries

Table D.1: Summary of object distribution per image and class (all annotations).

	Object count per image	Object count per class
nbr.val	8550.000	35.000
nbr.null	0.000	0.000
nbr.na	0.000	0.000
min	1.000	174.000
max	55.000	7920.000
range	54.000	7746.000
sum	51312.000	51312.000
median	4.000	625.000
mean	6.001	1466.057
SE.mean	0.066	326.056
CI.mean.0.95	0.129	662.626
var	37.268	3720947.467
std.dev	6.105	1928.976
coef.var	1.017	1.316

D.2.2 Graphical summaries

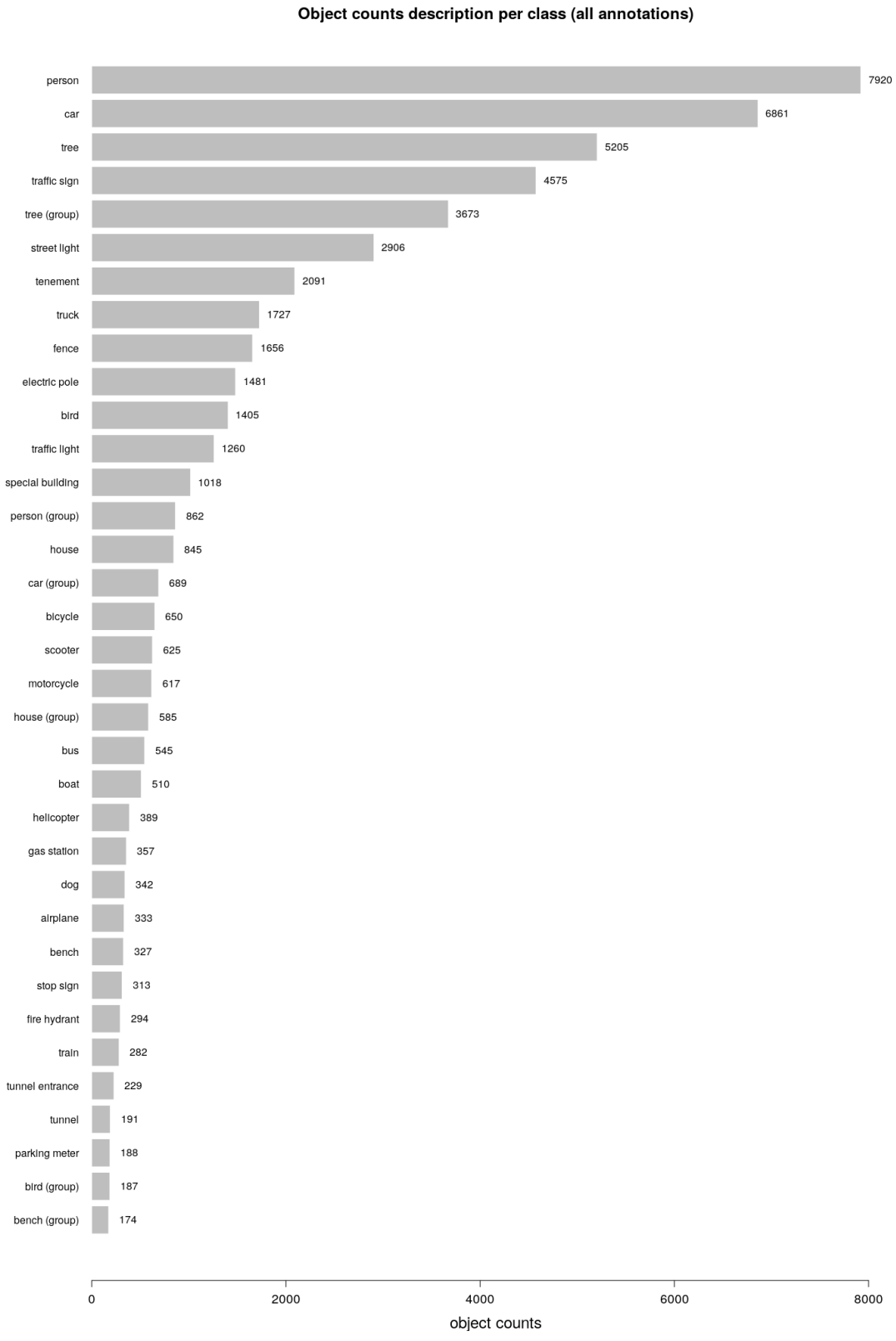


Figure D.2: Distribution of object classes on all annotations.

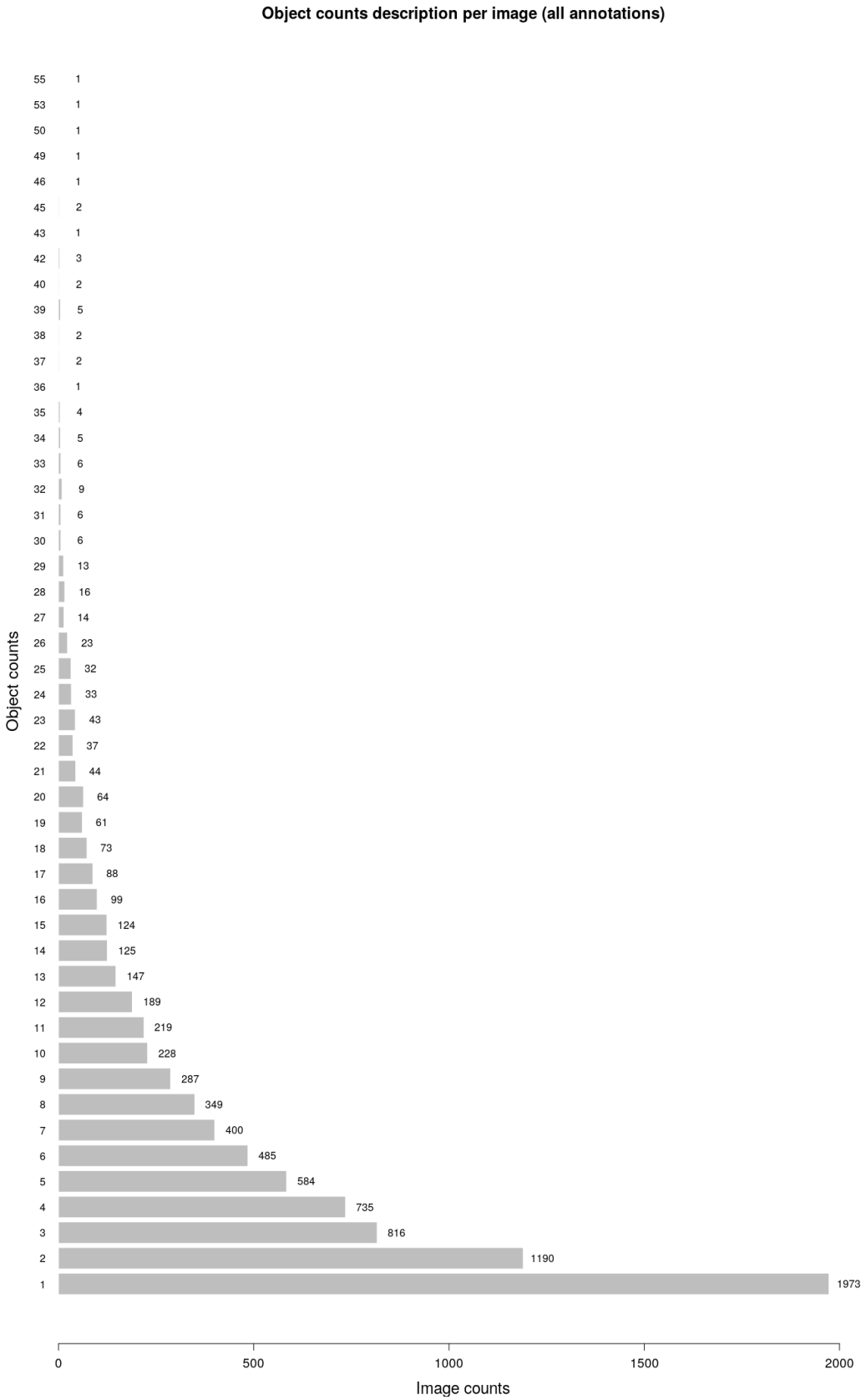


Figure D.3: Distribution of object frequencies on all annotations.

D.3 Description of training images

D.3.1 Numerical summaries

Table D.2: Summary of object distribution per image and class (training annotations).

	Object count per image	Object count per class
nbr.val	6800.000	35.000
nbr.null	0.000	0.000
nbr.na	0.000	0.000
min	1.000	110.000
max	55.000	5687.000
range	54.000	5577.000
sum	39680.000	39680.000
median	4.000	455.000
mean	5.835	1133.714
SE.mean	0.071	249.809
CI.mean.0.95	0.140	507.673
var	34.630	2184162.151
std.dev	5.885	1477.891
coef.var	1.008	1.304

D.3.2 Graphical summaries

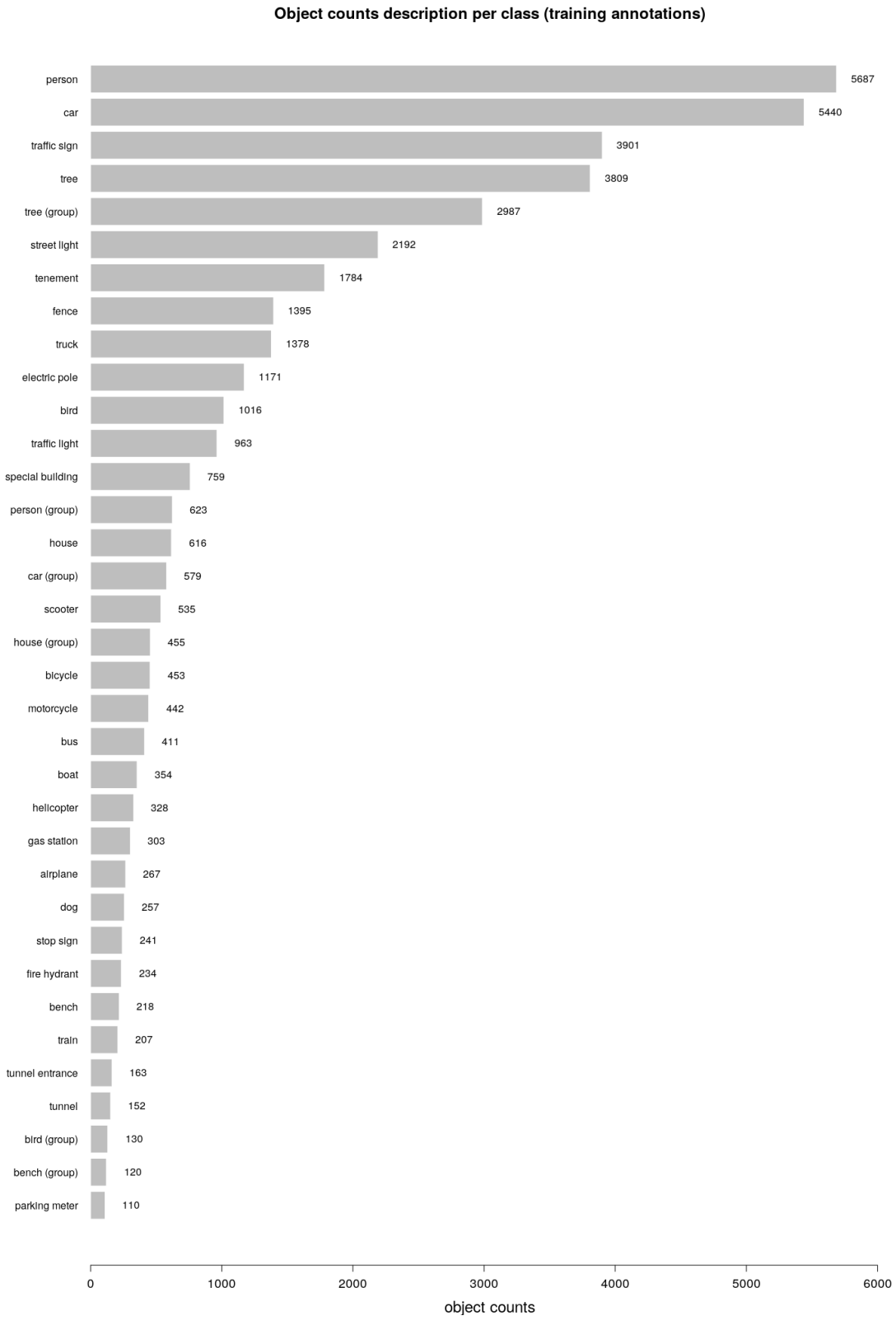


Figure D.4: Distribution of object classes on annotations associated with the training data.

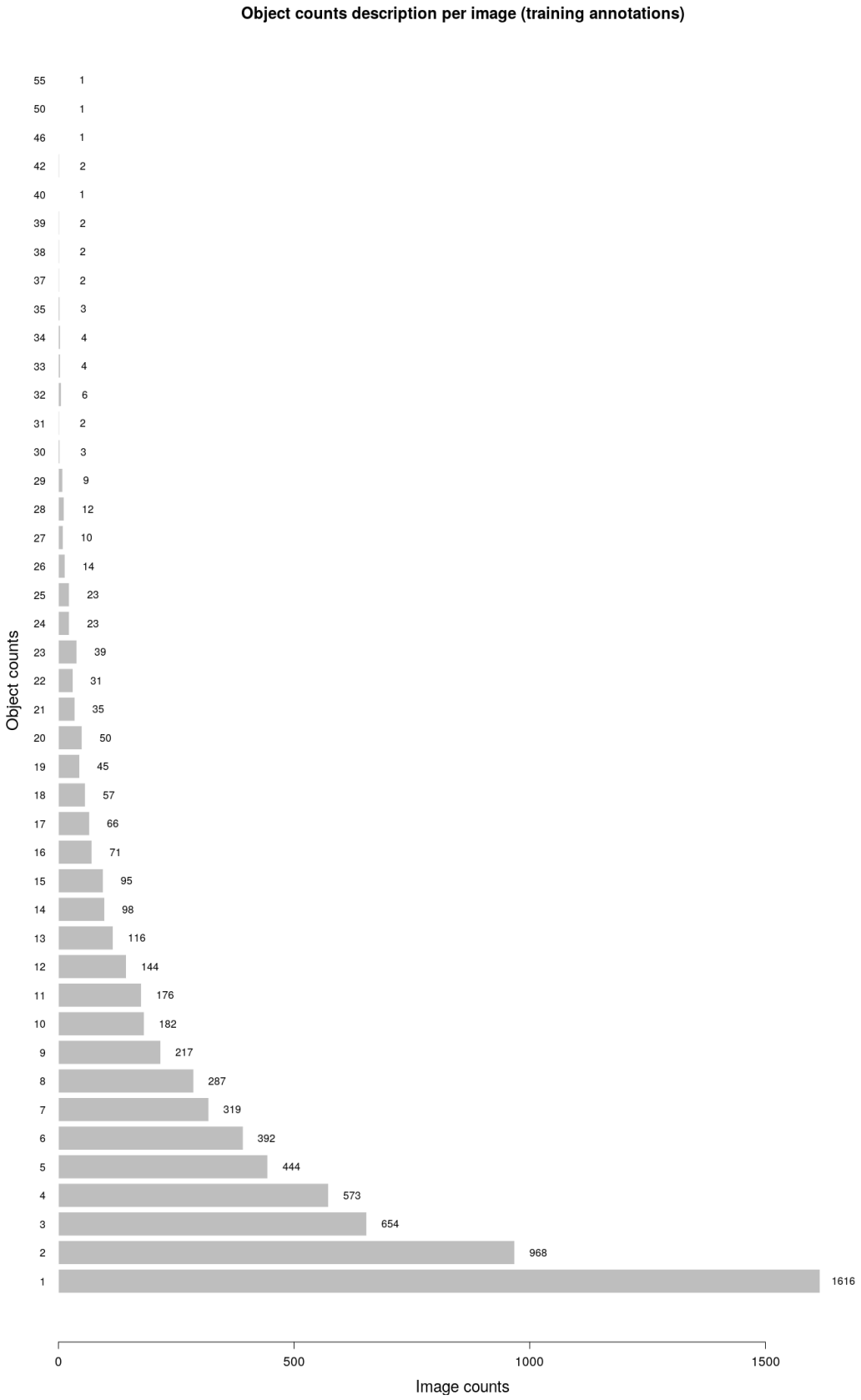


Figure D.5: Distribution of object frequencies on annotations associated with the training data.

D.4 Description of validation images

D.4.1 Numerical summaries

Table D.3: Summary of object distribution per image and class (validation annotations).

	Object count per image	Object count per class
nbr.val	1750.000	35.000
nbr.null	0.000	0.000
nbr.na	0.000	0.000
min	1.000	39.000
max	53.000	2233.000
range	52.000	2194.000
sum	11632.000	11632.000
median	4.000	156.000
mean	6.647	332.343
SE.mean	0.164	79.553
CI.mean.0.95	0.321	161.671
var	47.021	221503.114
std.dev	6.857	470.641
coef.var	1.032	1.416

D.4.2 Graphical summaries

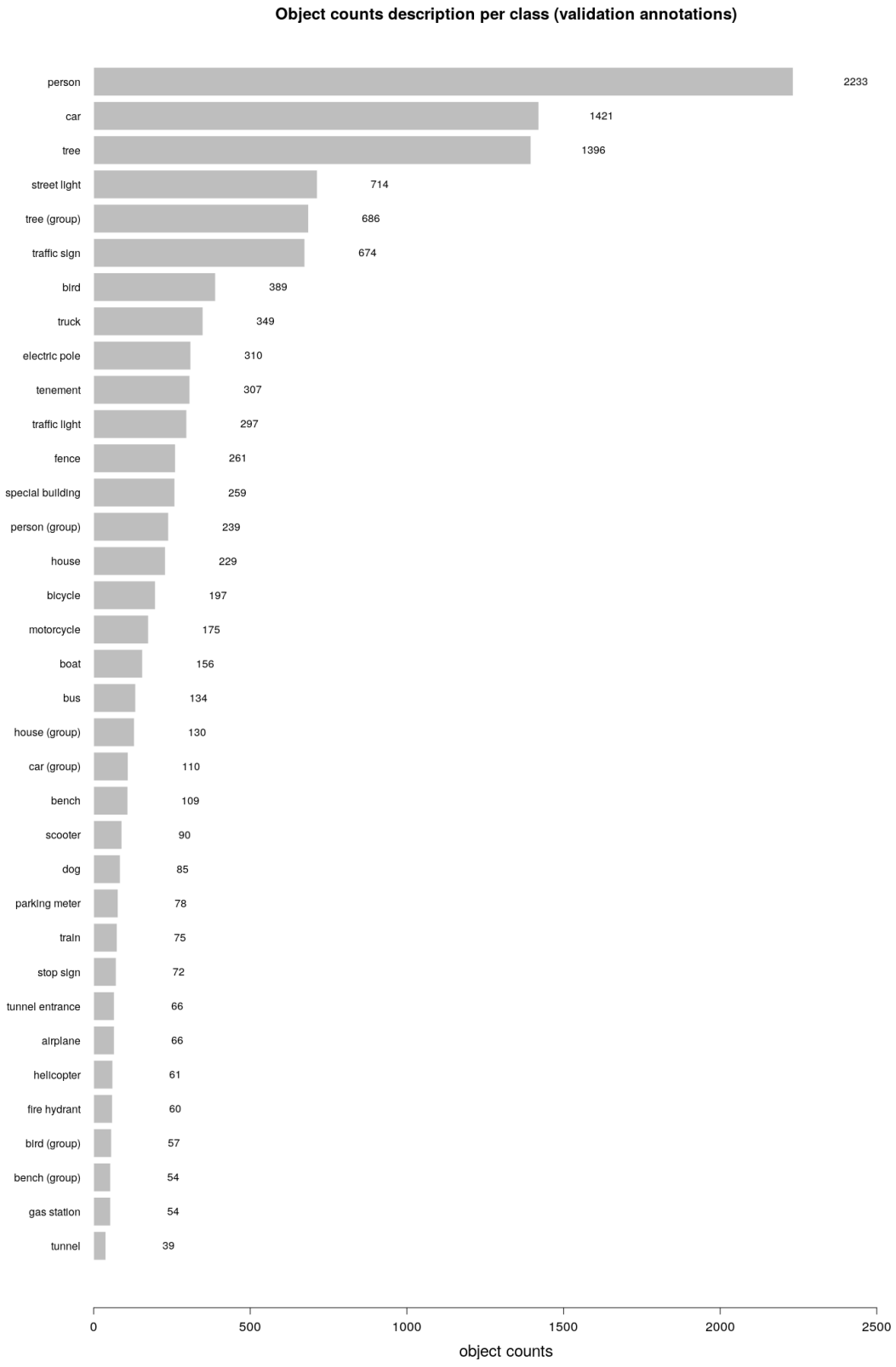


Figure D.6: Distribution of object classes on annotations associated with the validation data.

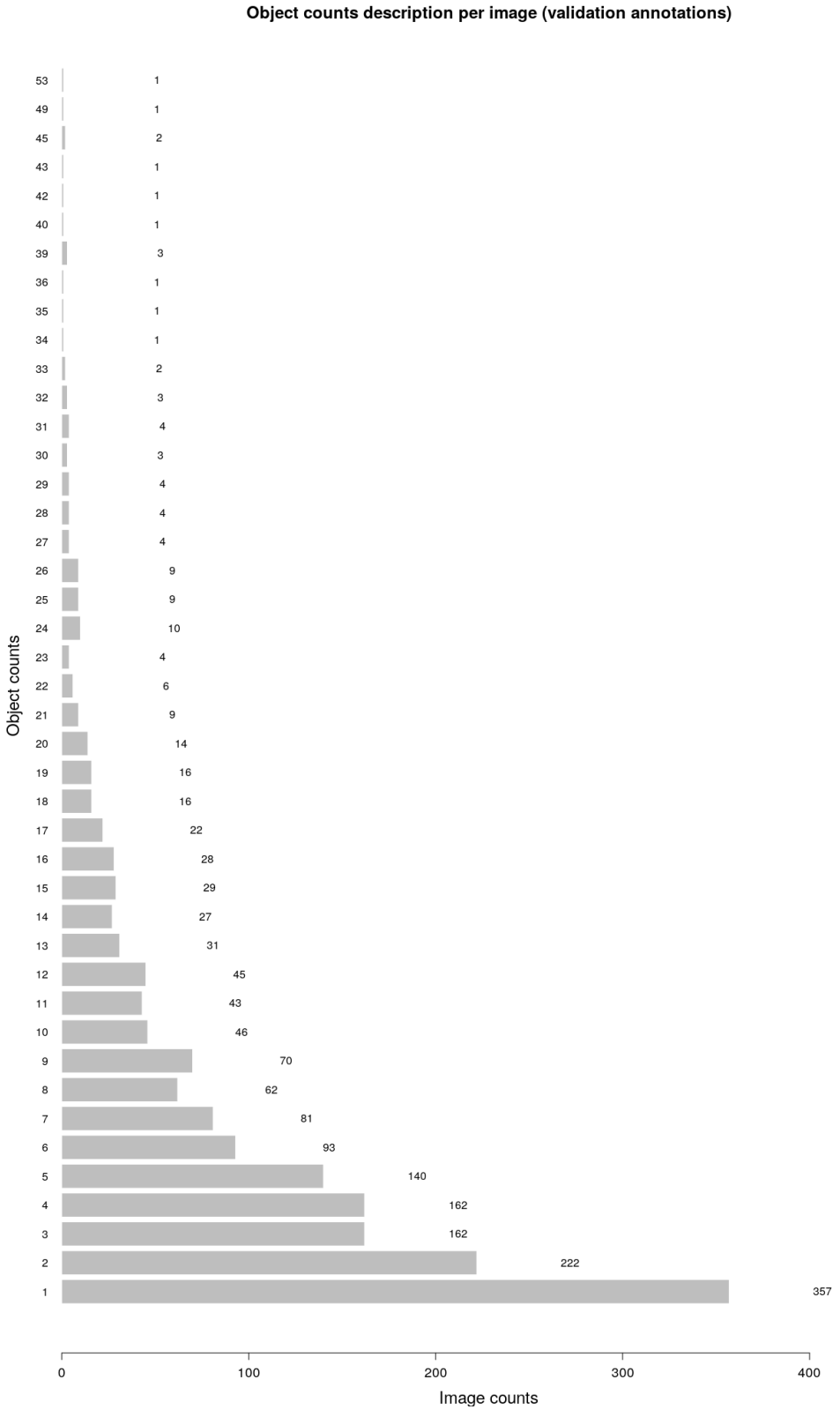


Figure D.7: Distribution of object frequencies on annotations associated with the validation data.

D.5 Performance metrics

D.5.1 Precision and recall

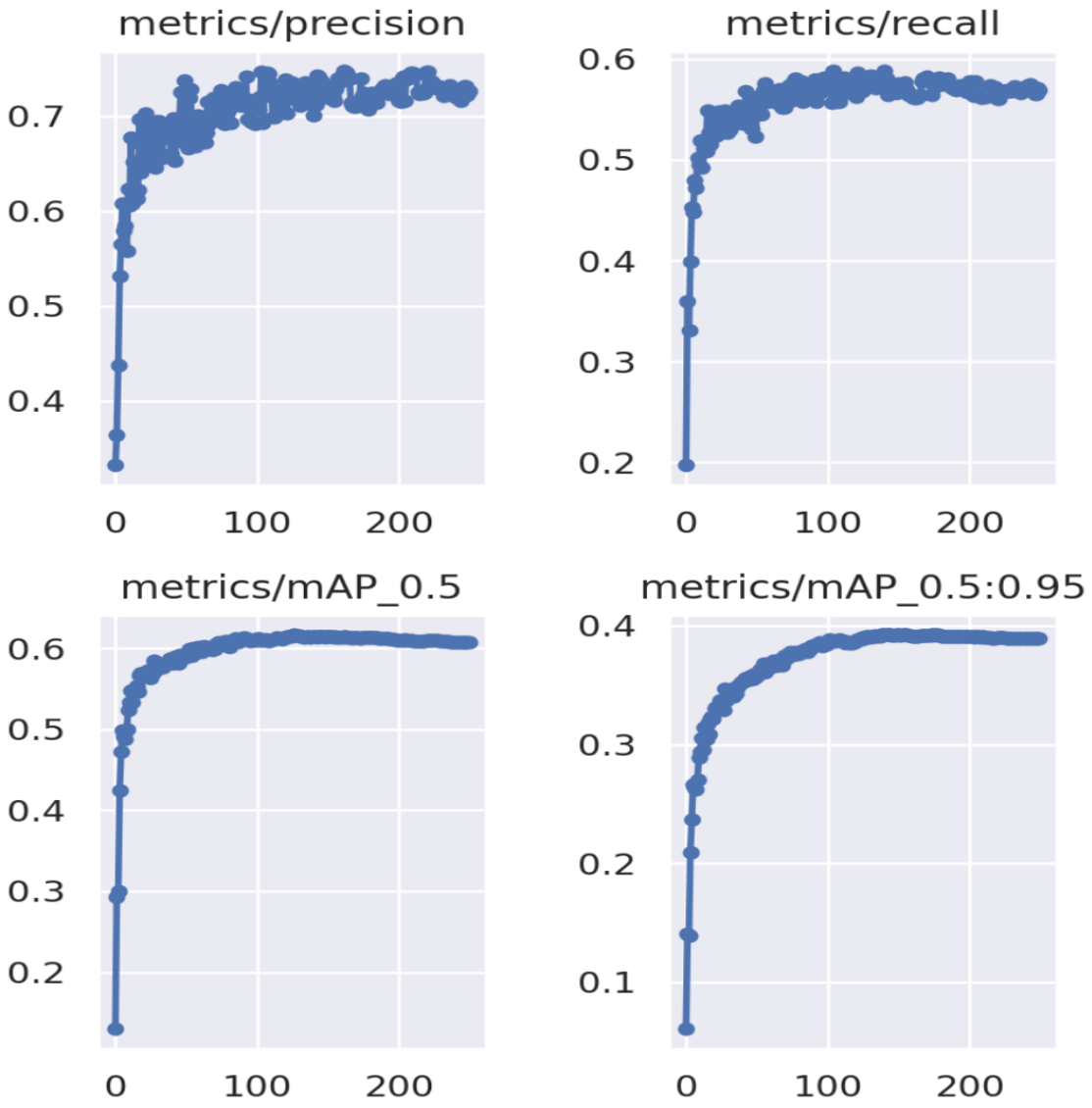


Figure D.8: Evaluation metrics plots for the custom traffic scene object detector based on the YOLOv5 model. Here, the metrics are related to precision and recall calculated on the validation data.

Class	Images	Labels	P	R	mAP_0.5	mAP_0.5:0.95
all	1750	11631	0.734	0.566	0.614	0.392
tree (group)	1750	686	0.648	0.511	0.567	0.314
airplane	1750	66	0.907	0.743	0.823	0.606
truck	1750	349	0.559	0.401	0.418	0.261
car	1750	1421	0.694	0.712	0.725	0.47
gas station	1750	54	0.98	0.963	0.97	0.656
tree	1750	1396	0.653	0.458	0.509	0.259
electric pole	1750	310	0.56	0.419	0.437	0.176
helicopter	1750	61	0.95	0.918	0.95	0.69
person	1750	2233	0.707	0.605	0.643	0.339
person (group)	1750	239	0.781	0.381	0.475	0.317
tenement	1750	307	0.566	0.459	0.416	0.224
bench (group)	1750	54	0.831	0.519	0.625	0.495
street light	1750	714	0.56	0.343	0.338	0.118
traffic sign	1750	674	0.746	0.691	0.708	0.47
fence	1750	261	0.642	0.398	0.418	0.25
traffic light	1750	296	0.723	0.608	0.65	0.305
house (group)	1750	130	0.79	0.362	0.463	0.251
tunnel	1750	39	0.918	0.86	0.938	0.7
bus	1750	134	0.631	0.604	0.598	0.437
house	1750	229	0.505	0.183	0.222	0.0984
bird	1750	389	0.613	0.411	0.443	0.309
parking meter	1750	78	0.772	0.607	0.683	0.492
tunnel entrance	1750	66	0.834	0.606	0.719	0.408
bird (group)	1750	57	0.869	0.466	0.629	0.394
bench	1750	109	0.658	0.431	0.464	0.303
special building	1750	259	0.676	0.432	0.487	0.312
car (group)	1750	110	0.772	0.522	0.573	0.326
fire hydrant	1750	60	0.846	0.783	0.821	0.606
motorcycle	1750	175	0.734	0.503	0.547	0.291
scooter	1750	90	0.838	0.811	0.899	0.607
stop sign	1750	72	0.883	0.819	0.873	0.716
train	1750	75	0.762	0.653	0.739	0.487
dog	1750	85	0.792	0.625	0.686	0.461
bicycle	1750	197	0.676	0.519	0.579	0.353
boat	1750	156	0.631	0.487	0.469	0.227

Figure D.9: Evaluation metrics numerical values per class of objects for the custom traffic scene object detector based on the YOLOv5 model. Here, the metrics are related to precision (P), mean average precision (mAP) and recall (R) calculated on the validation data. Precise that the second row displays these metrics with respect to all object classes.

D.5.2 Confusion matrix

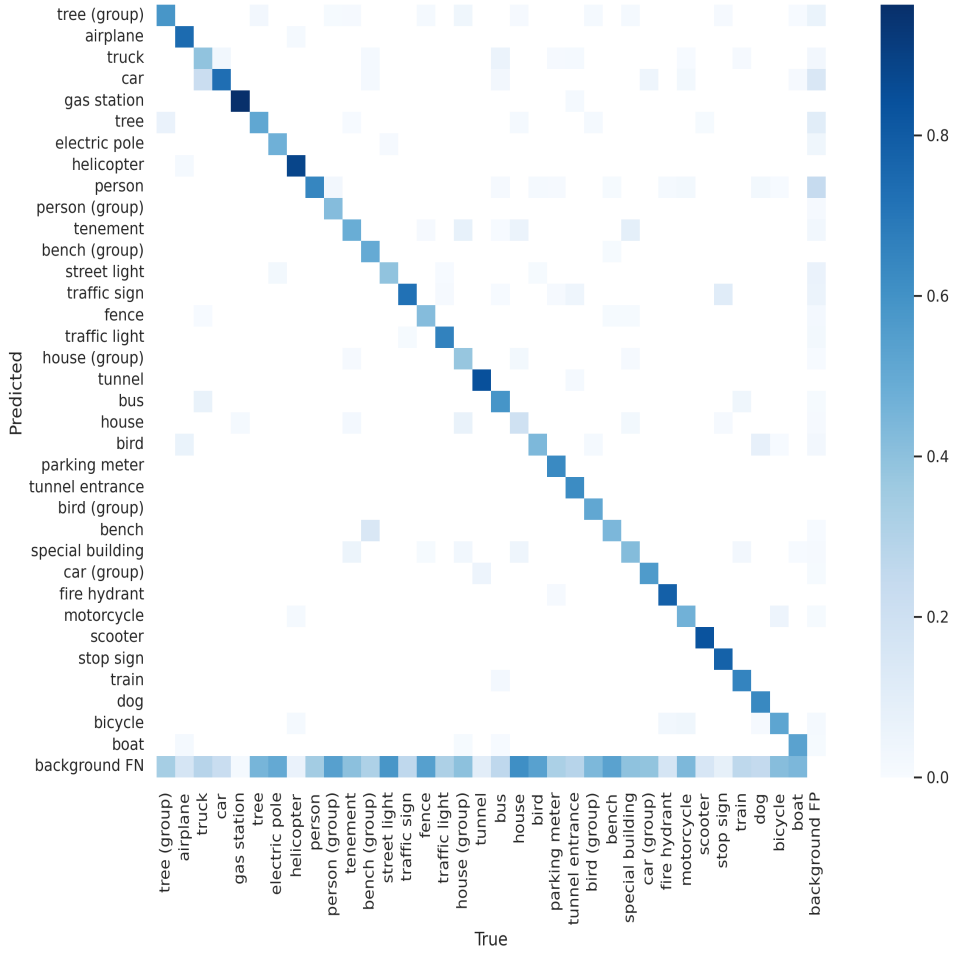


Figure D.10: Confusion matrix plots for the custom traffic scene object detector based on the YOLOv5 model. Here, the confusion matrix is estimated on the validation data.