CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Application of Machine Learning Algorithms for Post Processing of Reference Sensors

Utilization of Deep Learning Methods for Object Detection to Camera Data collected from vehicle's Reference Sensors

Master's thesis in Computer Science and Engineering

VASILIKI LAMPROUSI

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

Master's thesis 2021

Application of Machine Learning Algorithms for Post Processing of Reference Sensors

Utilization of Deep Learning Methods for Object Detection to Camera Data collected from vehicle's Reference Sensors

VASILIKI LAMPROUSI

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2021

Application of Machine Learning Algorithms for Post Processing of Reference Sensors
Utilization of Deep Learning Methods for Object Detection to Camera Data collected from vehicle's Reference Sensors
VASILIKI LAMPROUSI

Supervisor: Huu Le, Department of Electrical Engineering
Advisors: Georgia Diakou & Ricardo Silva, Volvo Car Corporation
Examiner: Christopher Zach, Department of Electrical Engineering

Master's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2021

Application of Machine Learning Algorithms for Post Processing of Reference Sensors
Utilization of Deep Learning Methods for Object Detection to Camera Data collected from vehicle's Reference Sensors
VASILIKI LAMPROUSI
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The Autonomous Drive (AD) systems and Advanced Driver Assistance Systems (ADAS) in the current and future generations of vehicles include a large number of sensors which are used to perceive the vehicle's surroundings. The production sensors of these vehicles are verified and validated against reference data that are originated from high-accurate reference sensors that are placed in a reference roof box at the top of the vehicle.

In this thesis, are explored ways to strengthen the reference camera data by applying deep machine learning algorithms together with other techniques for 2D object detection. For this reason, they are used two driving related datasets, public Berkeley DeepDrive dataset (BDD100K) and Volvo's annotated data. Also they are trained and evaluated two state of the art deep learning algorithms for object detection, Mask R-CNN [5] and YOLOv4 [25]. Finally, it is implemented in conjunction, a semi-supervised technique to improve the predictive performance using unlabeled data. The utilized semi-supervised learning framework is called STAC and it is introduced in the paper *A Simple Semi-Supervised Learning Framework for Object Detection* [27].

Keywords: Object detection, machine learning, camera, sensors, semi-supervised learning.

Acknowledgements

I would like to thank my academic supervisor Huu Le for the guidance, help and advice throughout the process of this project. I would also like to thank my company advisors Georgia Diakou and Ricardo Silva from Volvo Cars for all the support, guidance and help, for providing the access to Volvo's data and other good resources and helping with annotation. I would also like to thank Volvo's team that created the Volvo's annotation tool and let us use it and Ali Kadhim for helping with picking frames and annotating. Finally I would like to thank my family and my friends for their support.

Vasiliki Lamprousi, Gothenburg, February 2021

Contents

List of Figures

List of Tables

1
Introduction

With recent developments of artificial intelligence (AI), Autonomous Driving (AD) is become more popular and considered the future of smart transportation. An AD system allows the vehicle to be driven without human's intervention. In order to achieve self driving capabilities, it is crucial for any AD algorithms to perceive the surrounding environment, locate the vehicle's positions, as well as detect and recognize surrounding objects that may interfere with its movement, in order to assure the safe movement without human supervision. The safety of these vehicles is highly dependent on the performance of multiple sensors in order to make correct decisions. In order to achieve human-level perception, it is common for a modern autonomous vehicle to be equipped with a variety of high-quality sensors. These sensors include radars, cameras, Light Detection and Ranging (LiDAR) [1], differential GPS system, ultrasonic sensors and their output is used by sensor fusion. However, there remain some challenges that needs to be addressed for existing sensor systems. Particularly, they often operate in environments that are very noisy, while the underlying processing algorithms could be imperfect, hence even if the state-of-art sensors are installed, they may still produce wrong outputs, which could hamper the performance of the underlying AD systems. This project concerns the verification and validation of the sensors that are used for AD during every car development process. More specifically, the outcome of the project is a system that can provide correct reference (ground-truth) data to verify the performance of the installed sensors. These sensors are referred to as production sensors and are integrated around and inside the vehicle in order to have a clear picture of the surrounding world.

While there are many techniques to provide the ground-truth reference data, Volvo Car Corporation (VCC) would like to automate this process so that every single manufactured car can be automatically verified. Currently, in order to verify the production sensors, Volvo installs another sensor system which contain more sensors with high resolutions and better accuracy (compared to the production sensors). These sensors are referred to as reference sensors and are placed in a reference box on top of the vehicles. The reference box can be seen in Figure 1.1. The sensors that are integrated in the reference box are LiDAR, radar, camera, differential GPS system, and so on. Figure 1.2 shows the field of view of the four cameras that are placed on the reference box and the field of view of the LiDAR. The data of these sensors is referred to as reference data and is used to verify the production sensors placed in the vehicle. In the ideal case, the reference data is expected to provide ground-truth reference, so that the performance of production sensors can be evaluated. Therefore, the goal of the project is to generate correct ground-truth

reference data.

Figure 1.1: The reference box placed on the roof of the vehicle, which is used for the AD and ADAS development and verification.

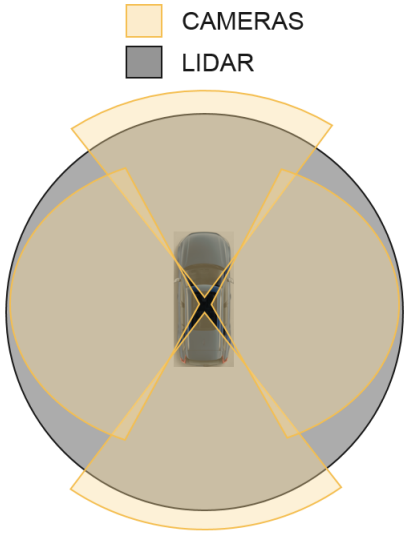


Figure 1.2: Field of view of the LiDAR and the four camera sensors that are integrated in the reference box.

Object detection is a computer vision technique that deals with detecting locations and assigning correct labels to the objects that are captured in an image or a video. This method has many applications in AD systems and Advanced Driver Assistance Systems (ADAS). Some examples are vehicle and pedestrian detection, lane and road edge detection, traffic signs/lights detection and so on.

This thesis will apply machine learning algorithms for 2D object detection to the

reference data from the front camera of the reference box. The thesis is conducted in cooperation with VCC. A fraction of the frames of video recordings of VCC reference cameras are annotated and used for training and testing.

1.1 Objective

As highlighted above, one important issue is the verification and validation of the production sensors integrated in vehicles that are used in the ADAS & AD systems. One way to validate these sensors is to use reference data from the reference box, post process them and compare their performance with the production sensors data. This post process could be done with the use of a huge amount of annotated (labelled) data from the cameras placed in the reference box. Annotating a huge amount of data, though, is time-consuming and expensive. The reference camera data that is provided from the reference box at the moment have incomplete labelling.

The research question is if it is possible to develop a trained network using an existing annotated dataset, and apply it to the data that we get from the reference box, and in return get high accurate object detection and classification. The current project focuses mainly on contributing to the high-level outputs from the reference camera sensors by materializing high accuracy object detection to these camera data. There is plenty of data that is collected and stored with these cameras and there are different methods that they can be applied to reach this goal.

1.2 Background and Motivation

Lately, by using deep neural network based algorithms, object classification, detection and semantic segmentation solutions are significantly improved [56, 10, 9]. Deep learning based object detection algorithms are divided into two categories, two-stage detectors and one-stage detectors. Two-stage detectors (Faster R-CNN [4], Mask R-CNN [5]) are known to have high localization and object recognition accuracy, whereas the one-stage detectors (YOLO [7], SSD [10]) are known to achieve high inference speed. The first stage of two-stage detectors, proposes candidate object bounding boxes and the second stage extracts features for the classification and bounding-box regression tasks. On the other hand, the one-stage detectors propose predicted boxes from input images directly without region proposal step [18]. The technology though evolves so fast and the newer two-stage detectors tend to be faster and the one-stage detectors tend to be more accurate.

The present project is aimed to improve the reference data from the reference camera sensors, to measure the production camera sensors performance. For this reason, this study focuses on a highly accurate detection of objects (cars, tracks/buses, pedestrians, motorcycles and bicycles). A lot of research has been done to the field of object detection for AD. Most of these studies focuses on real time speed detection [22, 57] or they solve 3D object detection problems [23, 58, 59] which is not our case.

On the other hand, a lot of advance has been made to highly accurate object detectors in general. Some of the state-of-the-art deep learning models for object detection are FPN [24], Mask R-CNN [5], RetinaNet [17] and YOLOv4 [25]. There are also important studies in semi-supervised learning in the object detection field that claim to be as accurate or even more than common supervised techniques [27, 60].

Mainly this project aims to apply machine learning algorithms for object detection on the camera data of the reference box after post processing the collected data. Right now reference data is LiDAR based only and cameras of the reference box are used mainly for visualization. In this project are explored different methods for object detection to the camera data that can give us the best performance. From the performance aspect, not only the accuracy of the neural network is required but also all the process of training and labeling the data are needed to be taken into consideration. Based on these facts, a novel method is investigated. In the future our results can be fused with the LiDAR output to have a better reference system.

1.3 Goals and Challenges

The overall encompassing goal for this thesis is to enhance the reference data of camera sensors on the reference box by applying deep learning methods for object detection. The detected objects are then used as reference data to validate the production sensors. To achieve this goal, we study the use of two state of the art object detectors and apply them to Volvo data. In addition, we also investigate the use of a semi-supervised learning approach that trains a network on labeled data and then improves it using unlabeled data.

Some of the challenges that are handled is the large amount of data which is not trivial and the different camera characteristics (resolution, color balance, focal length etc.) and scene characteristics (objects in different poses, different relative frequency of objects etc.) of the public and Volvo's dataset. For this reason it is tested image rescaling, color adjustments data augmentation and further training of the networks with Volvo's annotated data. Finally, data provided by camera sensors are usually noisy or contain missing/occluded regions. This is one more challenge that is handled.

1.4 Method outline

Various scientific approaches are appropriate for this challenge. In this thesis, they are used existing state-of-the-art algorithms, Mask R-CNN [5] and YOLOv4 [25]. These algorithms are trained with a public, driving dataset and then it is used transfer learning and the networks are trained more with annotated images from the reference cameras. Their performance is compared and presented. Finally it is used a semi-supervised technique that is presented in the paper *A Simple Semi-Supervised Learning Framework for Object Detection* [27]. Following this technique, a network is trained with labeled data and then it is trained in conjunction with

unlabeled data with pseudo labels. This last technique is implemented to view if there is any increase of the performance of the detector when it is given additional unannotated data.

2
Theory

2.1 Computer Vision

Computer Vision is the field of computer science that seeks to develop techniques that enable computers to gain high-level understanding of the visual world. Computer vision researchers focus on developing a wide range of visual perception algorithms for many practical tasks such as: (i) object recognition and classification, which aims to predict and determine the classes of the objects of interest that are present in an image, (ii) object detection in order to determine the location of the semantic objects of a given class that appear in an image, and (iii) image segmentation in order to translate an image into meaningful segments by classifying each pixel. In practice, several algorithms are often combined in one vision system to fulfill a specific task. All these computer vision problems are extremely challenging because they require the utilization of a broad range of mathematics and statistical models so that they can recover unknowns from insufficient amount of information to fully specify the solutions in the real world with the highest accuracy [62]. Some of the best-known computer vision techniques that are relevant to this project include image classification, object detection, semantic segmentation and instance segmentation. In addition, several other applications such as Structure from Motion (SfM) [80], Simultaneous Localization and Mapping (SLAM) [81] are also very popular in many autonomous driving systems.

Image Classification

Image classification aims to automatically classify images into predefined classes (as shown in Figure 2.1(a)). The great development of image classification occurred when the large-scale image dataset "ImageNet" [15] was created by Feifei Li in 2009. It contained 15 million images across 22000 classes of objects. At the same period, deep learning began to obtain great results in computer vision. Deep learning is a machine learning algorithm, which automatically extracts higher level features from the input with the use of neural networks (more than three layers) and incrementally boosts the achieved accuracy when given a huge training dataset. AlexNet [65] is classic convolutional neural network (CNN) architecture that represents a remarkable milestone in the modern history of neural network and won the first prize at the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012. Inspired by AlexNet, VGGNet [66] and GoogleNet [67] focus on designing deeper networks and improve further the accuracy. ResNet [26] was the winner of ILSVRC

in 2015 by proposing to use a shortcut connection between residual blocks to make full use of information from previous layers and keep the gradients during backward propagation. DenseNet [68] establishes connections between all previous layers and the current layer. SENet [69] proposes a "squeeze-and-excitation" (SE) unit by taking channel relationship into account. NASNet [70] adopts a neural architecture search (NAS) framework derived from reinforcement learning [71] and achieves the state-of-the-art accuracy on ImageNet.

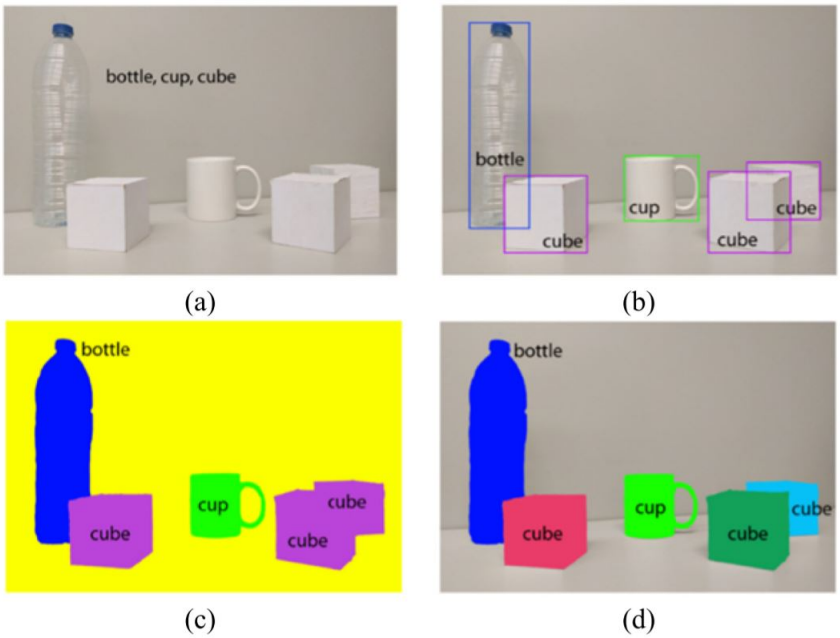


Figure 2.1: An example of different computer vision techniques: (a) image classification, (b) object detection, (c) semantic segmentation, and (d) instance segmentation [61].

Object Detection

Object detection is a computer vision technique which aim is to determine and locate the objects of interest in an image or a video (as shown in Figure 2.1(b)). Both object detection and image classification handle a large number of objects which in most cases differ a lot and are not easily recognised. However, object detection is more difficult than image classification, because it focuses on identifying the accurate location of the object of interest. A deeper description of this technique is presented in following sections.

Semantic & Instance Segmentation

Image segmentation is a pixel-level classification which divides an image into regions where an object or area of interest is represented. This is succeeded by classifying each pixel into a specific category. Image segmentation can be divided into two subbranches: (i) semantic segmentation, which aims to assign each pixel in an image to a semantic object class (as shown in Figure 2.1(c)), and (ii) instance segmentation,

which further improves semantic segmentation by predicting different labels for different instances of the same class (as shown in Figure 2.1(d)). Fully convolutional network (FCN) [72] is the forerunner framework for segmentation tasks that successfully implements pixel-wise dense predictions for semantic segmentation in an end-to-end CNN structure. FCN uses convolution layers instead of fully-connected layers that are used by most of the well-known architectures of classification tasks (e.g.VGG [66], GoogleNet [67], etc.). These layers can have inputs of varying sizes, and their output is a heatmap instead of a vector with classification scores [62]. Some of the state-of-the-art methods on this field are DeepLab [73], RefineNet [74] PSPNet [75] Mask-RCNN [5] and path aggregation network (PANet) [37].

2.2 Traditional Object Detection Models

The problem definition of object detection is to place a bounding box where the objects are located in the input image (object localization) and determine the category of the object in each box (object classification). Before the deep learning era, most research efforts focused on detecting a single class such as pedestrian [76, 77] and face [78] by designing a set of appropriate features (e.g. HOG [51], Harr-like [52], etc.). In these methods, object detection occurs by matching a number of predefined feature templates with each location in the image or the feature pyramids. Classifiers such as SVM [1] [28] and Adaboost [54] are often used for this purpose [62]. In general, the tree main steps that each traditional object detection model follows are: informative region selection, feature extraction and classification.

In the informative region selection step a multi-scale sliding window is used to scan the whole image so that all objects in any position, aspect ratio and size can be captured. In this way, all the possible positions of the objects can be found, but it is a computationally expensive method since there are produced many windows that are redundant. On the other hand, if the number of sliding window is restricted regions may be poor.

In the feature extraction step visual features are extracted to provide semantic and robust representation of different objects. Haar-like features [52], SIFT [50] and HOG [51] are the representative ones. It's difficult, though, to manually design a robust feature descriptor to describe all kinds of objects due to the diversity of appearances, illumination conditions and backgrounds.

Finally, in the classification step it is used a classifier to distinguish the category of an object from all the other categories. Commonly, support vector machines (SVM) [28] are used due to their good performance on small scale training data. Other options for the classification step is Deformable Part-based Model (DPM) [53] and AdaBoost [54].

However, during 2010-2012, small gains were obtained on PASCAL VOC object detection competition [16] based on these traditional methods. This showed the limitations of traditional detectors. More significant gain was obtained with the application of deep convolutional neural networks for object detection based on deep

[1] `Support Vector Machine (SVM) is an algorithm that finds a hyperplane that best separates data based on a set of features.`

learning techniques. Compared to traditional feature extractors, deep convolutional neural networks have deeper architectures having the ability to learn more complex features than the shallow ones [13].

2.3 Convolutional Neural Networks

Neural Networks are a set of algorithms that are trying to recognize underlying relationships in a set of data through a process that is similar to the way the human brain operates. Convolutional Neural Networks (CNNs) are a specialized kind of Neural Network that processes data that has a known grid-like topology like time-series data (1-D grid) and image data (2-D grid of pixels). The structure of CNNs is similar to regular Neural Networks, with trainable weights and biases, weighted sums over neuron inputs with computed outputs trough activation functions, and a problem specific loss function. The main difference between regular Deep Neural Networks and CNNs is two CNN-specific layers called convolution layers and pooling layers [57]. CNNs execute a mathematical operation called convolution which is a specialized type of linear operation. The convolution operation is the below:

$$s(t) = \int x(a)w(t-a)\,\mathrm{d}a, \tag{2.1}$$

where x and w are functions and w function is reversed and shifted. This operation is used to extract features from an image. A CNN architecture consists of Convolutional Layers, Pooling Layers and Fully Connected Layers.

Convolutional Layer

A convolutional layer is composed of neurons with learnable weights and biases. Each neuron in a convolutional layer receives inputs and calculates their outputs based on the learned weights and biases. The weights are visualized as matrices called filters or kernels. The convolutional operation (convolution) is executed by sliding the filter over the input in both directions, rows and columns. At every location, an element-wise multiplication is performed and summed together and the result is placed in the output feature map. In order to understand how convolutional layer operates, Figure 2.2 is displayed. Based on the Figure 2.2, we have an input image of 3×4 pixels, and a 2×2 2D convolution kernel. The kernel will execute 1 stride move from top left pixel to the bottom right pixel of the image. The kernel is a 2×2 matrix of weights (each component of the matrix is a weight). The result of all the convolutional operations is called feature map matrix and in our example it has 2×3 dimensions.
The stride specifies how much the filter shifts in each step when sliding through the input data. In our example the stride is one. When the stride increases, the output feature map is significantly reduced in size. An other option is to use padding. Padding is when additional columns and rows are added to enclose the input map with zeros. This increases the size of the input map and also it increases the performance, because it enables better extraction of information from the original borders

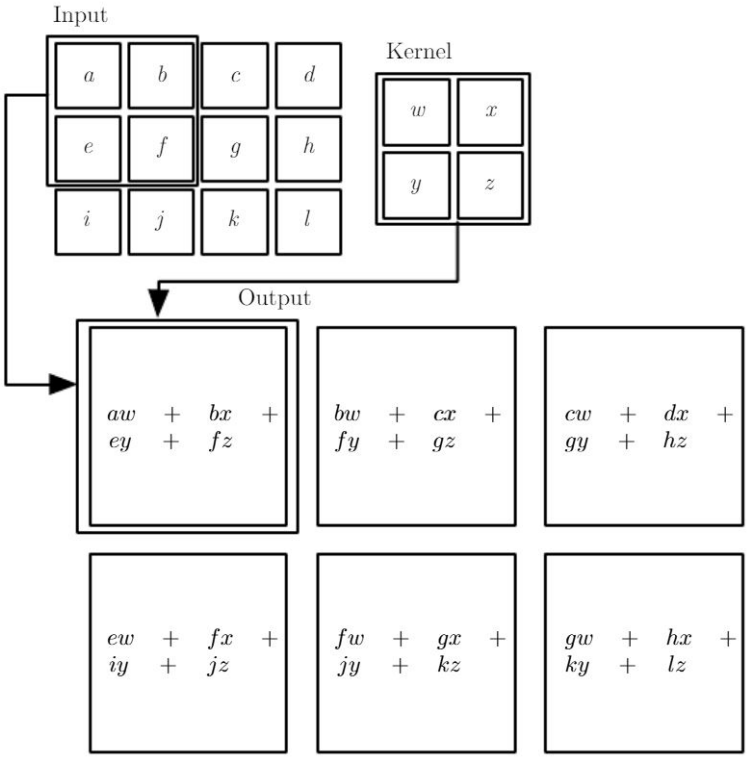


Figure 2.2: Example of 2-D convolution. The input image is 3×4 pixels, the kernel is 2D with dimensions 2×2 and the output feature map matrix has 2×3 dimensions since the kernel executes 1 stride[2].

of the input image. In the described example it was not used padding.

Pooling Layer

A pooling layer in a CNN can be perceived as a kind of down-sampling function. It is used to reduce the spatial size of the output feature map, hence it reduces the number of parameters and the computational complexity. It takes an activation map as input and it outputs a summary statistic of values from the input grid. Max pooling is one of the most often used types of pooling. This type splits the input feature map into equally sized regions and only the maximum value present in each region is kept as output. Two examples of pooling layers is presented in Figure 2.3

Fully Connected Layers

After features have been learned from convolutional and pooling layers, the reasoning from the features can be done through fully connected layers. The use of fully connected layers relies on the type of output aimed for, because it allows the movement from a grid representation to single values. This is typically useful when performing classification or regression based on the input as a whole. The first fully connected layer takes the output of the previous feature analysis layer and turns it

[2] `http://www.deeplearningbook.org/contents/convnets.html`

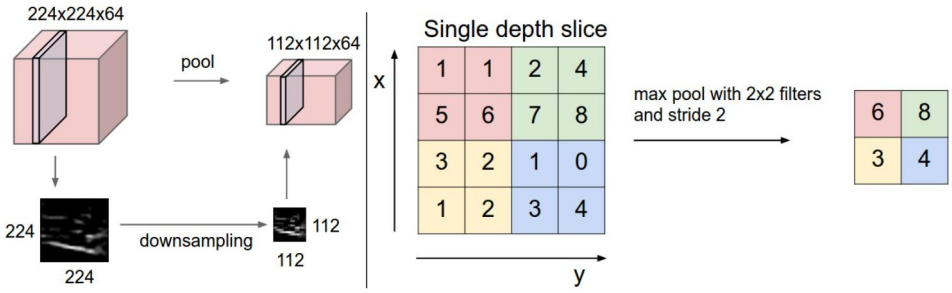


Figure 2.3: Left: A pooling layer with filter size 2 and stride 2 downsamples the input volume of size [224x224x64] into output volume of size [112x112x64]. Right: A max pooling example of stride 2 takes the max over 4 numbers of each 2x2 colored squares [79].

into a single vector ("flattens" the output) so that it can be an input for the next stage. It moves each individual feature map matrix value into a vector where each position in the vector is interpreted as an input value to the following fully connected layer. Then the first fully connected layer applies weights to predict the correct label. Finally the fully connected output layer gives the final output (classification or regression).

2.4 Deep Learning based Object Detection Models

The deep learning methods for object detection are mainly categorized into two types, two-stage detection and one-stage detection. The first type, initially generates region proposals in the image, regions where a possible object may be and then it classifies each proposal as a background or an object with specific label (category). The second type doesn't have a separately region proposal stage. It considers object detection as a regression or classification problem. It divides the original image roughly into a 2D grid and then each grid cell is used as a rough region to predict categories and locations.

Some of the most well-known two-stage detection methods are R-CNN [2], SPPNet [12], Fast R-CNN [3], Faster R-CNN [4], R-FCN [6] , FPN [24] and Mask R-CNN [5]. Some of these algorithms are correlated with each other, for instance Faster R-CNN improves Fast R-CNN by adding a region proposal network (RPN) to generate region proposals. The one-stage detection methods include YOLO [7], SSD [10], YOLOv2 [8], RetinaNet [17], YOLOv3 [9] and YOLOv4 [25]. These two pipelines are correlated by the anchors introduced in Faster R-CNN [13]. Two-stage detectors, in general, have high localization and object recognition accuracy, whereas the one-stage detectors achieve high inference speed.

2.4.1 Two-stage Detection

The first stage of two-stage detectors, proposes candidate object bounding boxes and the second stage extracts features for the classification and bounding-box regression

tasks. Region based CNN (R-CNN) family belongs to two-stage method.

2.4.1.1 R-CNN

The flowchart of R-CNN can be divided into the following three stages:

- Finding regions in the image that might contain an object. These regions are called region proposals.
- Extracting CNN features from the region proposals.
- Classifying the objects using the extracted features.

These steps can be shown in Figure 2.4.

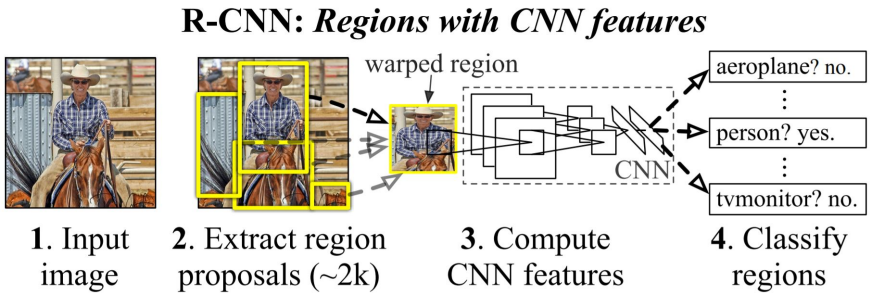


Figure 2.4: The stages of R-CNN [2] algorithm: 1) takes as an input an image, 2) extracts 2000 bottom-up region proposals, 3) computes features for each proposal using a CNN, and 4) classifies each region with class-specific linear SVMs.

The R-CNN [2] method generates about 2k region proposals via Selective Search [14] for each image. Selective Search is a region proposal algorithm that uses hierarchical and complementary grouping strategies based on size, color, texture, and shape compatibility to generate a small set of high-quality object locations (regions). Each region proposal is rescaled to a fixed size image and fed into a CNN trained on ImageNet [15] to extract features. Then, linear Support Vector Machine (SVM) classifiers are used to predict if there is an object in each region or if there is only background and classify the possible object. R-CNN yield a significant improvement on Pascal VOC07 dataset [16] on mean Average Precision (mAP) (from 33.7% to 58.5%). Mean average precision is described in §3.3.1.3. The main weakness of this model is the extremely slow detection speed (14s per image with GPU).

2.4.1.2 SPPNet

SPPNet [12] method overcomes the slow detection speed problem by introducing the Spatial Pyramid Pooling (SPP) layer (see Figure 2.5). This layer is placed on top of the last convolutional layer. It pools the features from the previous layer and generates a fixed-length output regardless of the size of the image or the region of interest. The output of SPP layer is fed into the fully connected layers. The feature maps are computed from the entire image only once, and then fixed-length representations of arbitrary regions are generated for training the detectors. This avoids repeatedly computing the convolutional features. SPPNet is almost 20 times faster than R-CNN and has as high accuracy (VOC07 mAP=59.2%). The two drawbacks of SPPNet are: 1) the training is still multi-stage (feature extraction

stage, network fine-tuning stage, SVM training and bounding box regressor fitting), 2) it only fine-tunes the fully connected layers and ignores all previous layers. The latter can result to an accuracy drop of very deep networks.

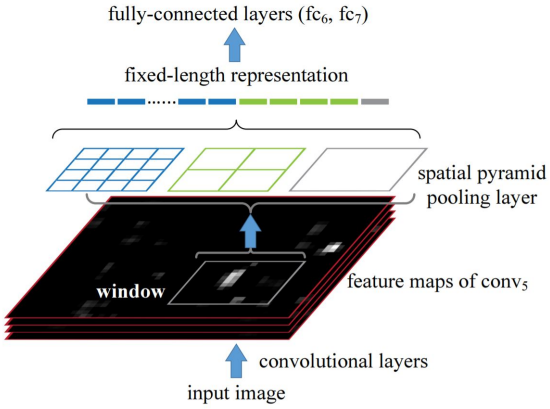


Figure 2.5: The architecture of SPPNet method [12].

2.4.1.3 Fast R-CNN

Fast R-CNN detector [3] improves further the R-CNN and SPPNet. Fast R-CNN enables the simultaneously training of a detector and a bounding box regressor by shared convolutional features. The whole image is processed with convolutional layers to produce feature maps, as it was in SPP-net. Then, a fixed-length feature vector is extracted from each region proposal with a region of interest (RoI) pooling layer, using Selective Search [14]. The RoI pooling layer is a case of SPP layer with only one pyramid level. Each feature vector is fed into a sequence of fully connected layers and the output is softmax probabilities and bounding-box regression offsets for each RoI. In Fast R-CNN method it is used multi-task loss function that jointly trains classification and bounding-box regression. Fast R-CNN increased the mAP to 70.0% for VOC07 and also icreased the detection speed over 200 times more than R-CNN. The architecture of Fast R-CNN is shown in figure 2.6.

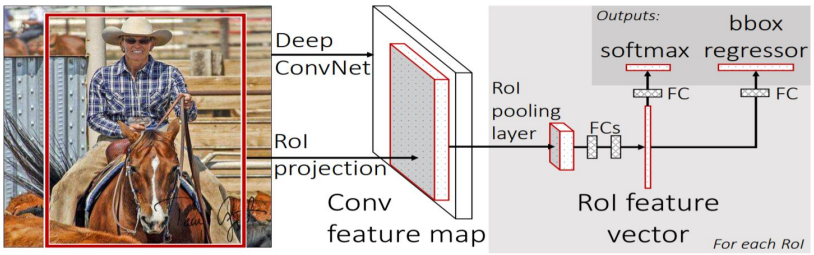


Figure 2.6: The architecture of Fast R-CNN method [3].

2.4.1.4 Faster R-CNN

Faster R-CNN [4] breaks through the speed bottleneck of Fast R-CNN by replacing Selective Search, which was used for generating region proposals, with the Region

Proposal Network (RPN). Convolutional feature maps that were used by region-based detectors, like Fast R-CNN, are used for generating region proposals as well in this model. On top of these convolutional features, Region Proposal Networks (RPNs) are constructed.

The Region Proposal Network, takes an image of any size as input and outputs a set of object proposals (rectangular), each with a score that measures membership to a set of object classes versus background that is called objectness score. This idea is modeled with a fully-convolutional network.

In particular, a small network slides over the convolutional feature map of the last shared convolutional layer and it generates the region proposals. This network is fully connected to a $n \times n$ spatial window of the input convolutional feature map. Each sliding window is mapped to a fixed lower-dimensional vector and then it is fed into two sibling fully-connected layers, a box-regression layer and a box-classification layer. This architecture is naturally implemented with $n \times n$ convolutional layer followed by two sibling 1×1 convolutional layers (for regression and classification). ReLUs[3] are applied to the output of the $n \times n$ convolutional layer.

In Figure 2.7 it is illustrated the Region Proposal Network. As it is shown in Figure 2.7, at each location of the sliding window, k region proposals are suggested. The k proposals are relative to the k reference boxes and are called anchors. Each anchor is centered at the corresponding sliding window, and has different scale and aspect ratio. In Figure 2.7 are used 3 scales and 3 aspect ratios, so $k = 9$ anchors at each position.

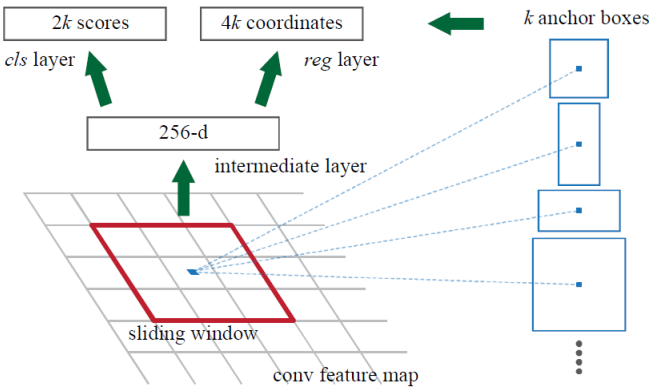


Figure 2.7: Region Proposal Network (RPN) [4].

Faster R-CNN is the first end-to-end, and the first near-realtime deep learning detector. Faster R-CNN increased the mAP of VOC07 to 73.2% and achieved mAP@.5=42.7% and mAP@[.5,.95]=21.9% to COCO dataset [35]. Region Proposal Network enables nearly cost-free region proposals. The architecture of Faster R-CNN is shown in Figure 2.8.

[3]The rectified linear activation function or ReLU is a piecewise linear function that outputs the input directly if it is positive, otherwise, it outputs zero. It is a common activation function for many types of neural networks.

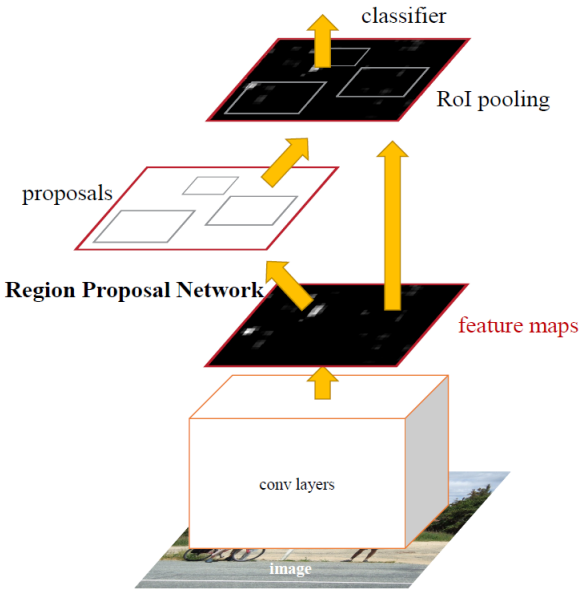


Figure 2.8: The architecture of Faster R-CNN method [4].

2.4.1.5 FPN

Before Feature Pyramid Network (FPN) [24], most of the deep learning based detectors run detection only to the network's top layer. The features in deeper layers of a CNN are known to be useful for class recognition but they are not so contributory to localizing objects. Hence, FPN introduces a topdown architecture with lateral connections, for building high-level semantics. A CNN naturally forms a feature pyramid through its forward propagation, so the FPN shows great advances for detecting objects with a wide variety of scales. FPN in a Faster R-CNN model achieves state-of-the-art results on the MS COCO dataset [35] (mAP@.5=59.1%, mAP@[.5, .95]=36.2%). FPN is a basic building block of many latest detectors.

2.4.2 One-stage Detection

The one-stage detectors directly predict object bounding boxes for an image without intermediate task. They pre-define a set of boxes to look for objects, then they use convolutional feature maps to predict class scores and bounding boxes. One-stage detectors are usually time efficient and can be used for real-time devices, but they sometimes struggle to adapt to arbitrary tasks (such as mask prediction).

2.4.2.1 YOLO

The first one-stage detector introduced was YOLO [7] and is the abbreviation of "You Only Look Once". It applies a single neural network to the full input image. This network divides the image into an $S \times S$ grid, it predicts a specific number of bounding boxes and confidence for each grid cell and it calculates the probabilities for each class in each of those boxes simultaneously. The confidence score of a bounding box is calculated by multiplying the class probability with the corresponding IoU

(see Section 2.6) between the predicted box and the actual box. Figure 2.9 illustrates the above steps. YOLO divides the input image into an $S \times S$ grid and predicts B bounding boxes for each grid, confidence for each box, and class probabilities. The final layer of the network outputs an $S \times S \times (C + B \times 5)$ tensor. The five in the equation corresponds to the x-coordinate for bounding box center, y-coordinate for bounding box center, bounding box width, bounding box height, and prediction confidence score.

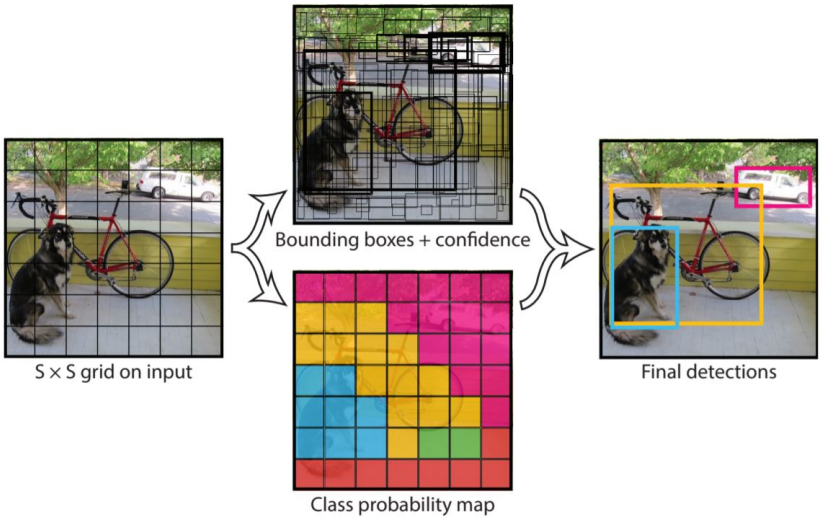


Figure 2.9: YOLO algorithm divides the input image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and class probabilities [7].

The architecture of YOLO method was inspired by the GoogLeNet model [67]. YOLO has 24 convolutional layers that extract features from the input image and they are followed by 2 fully connected layers that predict the output probabilities and bounding box coordinates. The network architecture of YOLO is shown in Figure 2.10.

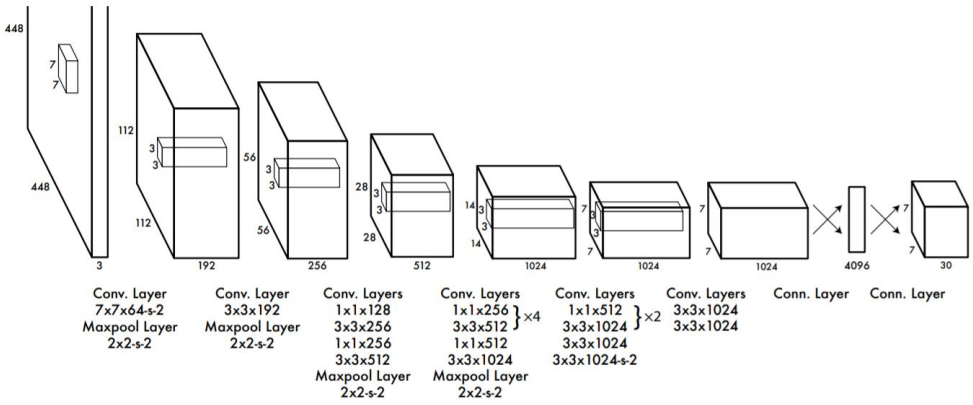


Figure 2.10: The architecture of YOLO consists of 24 convolutional layers and 2 fully connected layers [7].

The main contribution of YOLO is real-time detection. Also, YOLO handles detection as a regression problem, so a unified architecture extracts features from input images straightly to predict bounding boxes and class probabilities. However, YOLO's downsides are that it has worse localization accuracy than two-stage detectors and it gives unsuccessful detection of small objects.

2.4.2.2 SSD

Single Shot MultiBox Detector (SSD) [10] introduced the multi-reference and multi-resolution detection techniques, which improved the detection accuracy of one-stage detectors, especially for small objects. The main idea of multi-reference detection is to pre-define a set of reference boxes (anchor boxes) with different aspect-ratios and sizes at different locations in an image, and then predict the detection box based on these references. Multi-resolution detection, on the other hand, is a technique that detects objects of different scales at different layers of the network. Before SSD, detectors only run detection on their top layers. Multi-reference and multi-resolution detection is used by most of the state of the art object detection systems. SSD improves both the speed and the accuracy of the detection.

2.4.2.3 YOLOv2

YOLOv2 [8] is an improved version of YOLO, which adopts a plethora of ideas from past works with novel concepts and significantly improves YOLO's speed and precision. The techniques that improved YOLO are namely: Batch Normalization [55], high resolution classifier, convolutional with anchor boxes, predicting the size and aspect ratio of anchor boxes using dimension clusters, fine-grained features, multi-scale training and a custom deep architecture Darknet19.

2.4.2.4 YOLOv3

YOLOv3 [9] is an improved version of YOLOv2. YOLOv3 algorithm uses multi-label classification to adapt to more complex datasets containing many overlapping labels. Additionally, it utilizes three different scale feature maps to predict the bounding box. The last convolutional layer outputs a 3-d tensor with class predictions, objectness, and bounding box. Finally, YOLOv3 proposes a deeper and robust feature extractor, called Darknet-53, inspired by ResNet [26].

2.5 Semi-Supervised Learning

The progress made on object detection is mainly on training a stronger or faster object detector given sufficient amount of annotated data. There are cases though where it is hard to manually produce a sufficient number of annotated images. In such situations it is used a semi-supervised learning approach for object detection where the detector is improved by using unlabeled training data. In the subsections that follow, there are described three methods within semi-supervised learning that are utilized in the semi-supervised framework that is used in this report, called Consistency Regularization, Pseudo-Labeling and Noisy Student Training.

2.5.1 Consistency Regularization

Many recent state-of-the-art semi-supervised learning algorithms use consistency regularisation technique. This technique utilizes unlabeled data by relying on the assumption that the model should be invariant to perturbations happen on the same unlabelled image. In semi-supervised learning the perturbations have typically been based on image augmentations [30] [31]. Image augmentation is a technique which artificially creates images with different ways of processing, with rotation, shifts, flips and more.

2.5.2 Pseudo-Labeling

Pseudo-Labelling is a semi-supervised technique that has as main idea that the model itself should be used to obtain artificial labels for unlabeled data. Its initial motivation derives from entropy minimization, to encourage the network to perform confident predictions on unlabelled data [30] [31].

2.5.3 Noisy Student Training

Noisy Student Training [29] is a semi-supervised learning approach, which is based on the student-teacher framework. In this framework, a teacher generates targets that a student uses to train. In particular, Noisy Student Training has three main steps: 1) train a teacher model on labeled images, 2) use the teacher to generate pseudo labels on unlabeled images, and 3) train a student model on the labeled images and pseudo labeled images. This algorithm is iterated by putting back the student as the teacher and relabel the unlabeled data. When training the student, it is applied noise (e.g. dropout, stochastic depth, data augmentation via RandAugment) to make the student generalize better than the teacher.

2.6 Evaluation Metrics for Object Detection

Object detection models generally have a vary amount of predictions depending on the input image. This varying amount of outputs implies non-trivial evaluation.

Intersection over Union (IoU)

The Intersection over Union (IoU) is the ratio of the area of the intersection of the predicted bounding box and the ground truth bounding box and the area of the union of the two bounding boxes (see Figure 2.11). A perfect bounding box prediction has IoU = 1. It is common to have as a threshold for a positive prediction an IoU that is greater than 0.5 (they overlap by 50% or more). However, each dataset has its definition of what is a true positive prediction.

Possible scenarios of object detection prediction

Generally, there are four possible scenarios of predictions that can be made from an object detector.

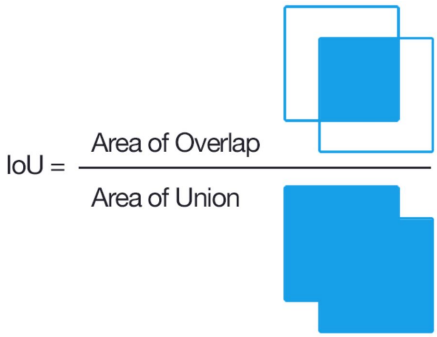


Figure 2.11: Graphical explanation of Intersection over Union[4].

- True positive (TP): IoU over the threshold (e.g >0.5) with the correct classification.
- True negative (TN): A correct prediction of background (no bounding box).
- False positive (FP): A predicted bounding box that does not match with any ground truth object or an IoU less than the threshold (or an additional overlapping prediction).
- False negative (FN): When there is no detection at all of an existing object or it detects wrong object category.

mean Average Precision (mAP)

Precision and recall are two commonly used metric to measure the performance of a given classification model. Precision refers to the percentage of the correctly predicted bounding boxes out of all bounding boxes predicted. Recall refers to the percentage of the correctly predicted bounding boxes out of all objects in the photo.

$$Precision = \frac{TP}{TP + FP} \tag{2.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.3}$$

The more predictions are made the recall percentage increases, but precision drops or becomes erratic as false positive predictions are made. The recall (x-axis) is plotted against the precision (y-axis) for each number of predictions to create a curve or line. The value of each point on this line is maximized (Interpolated Precision). The Interpolated Precision for a given Recall Value (r) is:

$$p_{interpolated}(r) = \max_{r' \geq r} p(r') \tag{2.4}$$

The area under the interpolated Precision-Recall curve is the Average Precision (AP) value for the class. There are variations on how AP is calculated. PASCAL VOC dataset [16] and MS COCO dataset [35] calculates it in a different way. The mean of the average precision (AP) of the images in a dataset is called the mean average precision, or mAP.

[4] https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

2.7 Transfer Learning

Transfer learning is a machine learning technique where a model trained on one task is exploited to improve generalization in another related task. In particular, a base network is firstly trained on a base dataset and task, and then the learned features are transferred to a second target network and are trained on a target dataset and task. This technique works better when the features are suitable to both base and target tasks and not only to the base task [48].

There are two transfer learning approaches, develop model approach and pre-trained model approach. In the first approach it is used an abundance of data to train the network and then all or parts of this model is used by the model of the second task as a starting point. The final model may need to be adapted or refined on data available for the task of interest. In the second approach it is chosen a pre-trained source model from released models created from large and challenging datasets. This pre-trained model is used as the starting point for the second task of interest. The model is then trained more on the data of the second task. The pre-trained model approach is common in the field of deep learning.

There are three possible benefits when transfer learning is used and they are illustrated in Figure 2.12. The first one is that the initial performance in the target task using only the transferred knowledge is higher compared to how it otherwise would be. The second possible benefit is the small amount of time that is needed when using transfer learning compared to the amount of time is needed when learning from scratch. Finally, it is the higher final performance level achievable in the target task when transfer learning is used. Ideally, all three benefits can be seen from a successful application of transfer learning [49].

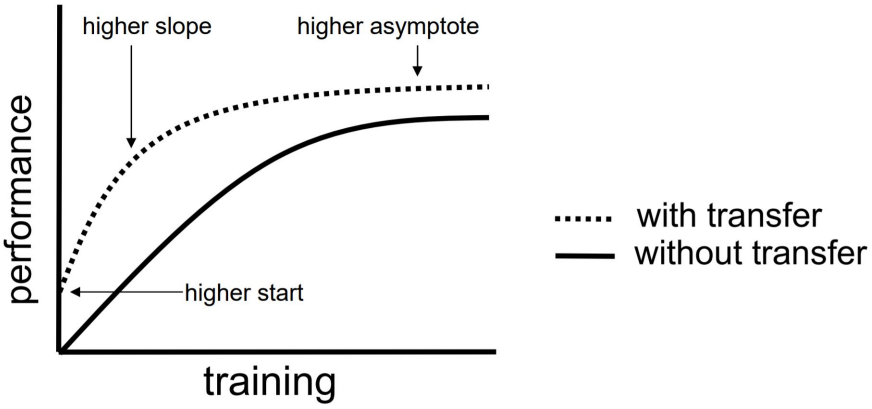


Figure 2.12: Three possible benefits of using transfer learning [49].

Transfer learning is a really useful technique especially when there is not much data available. This technique can enable to develop skillful models that they could not be developed otherwise.

3
Methods

Convolutional Neural Networks has pushed the limits of what was possible in the domain of image processing. The ability of deep learning techniques to learn feature representations automatically from data resulted to major improvements in object detection [32, 33]. For this reason all methods that are used in this project are within the deep learning field.

In the present project two methods for training object detection models in a supervised fashion are developed and evaluated. The utilized methods are Mask R-CNN [5] and YOLOv4 [25]. In both cases it is used transfer learning to succeed a better and faster result.

Mask R-CNN was introduced in the 2017 paper titled "Mask R-CNN" [5] and was revised on 2018. It is one of the state-of-the-art approaches for object recognition tasks. It is flexible, simple to train, easy to generalize to other tasks, gives top object detection results and won the Best Paper Award (Marr Prize) at the 16th International Conference on Computer vision (ICCV) 2017. We decided to use Mask R-CNN as it is one of the most representative two stage region-based CNN object detection algorithm and has state-of-the-art results on MS COCO dataset [35].

On the other hand, YOLOv4 was published in April 2020 and it is a significant upgrade, compared to YOLOv3, in terms of performance and speed. The architecture of YOLOv4 algorithm as well as some optimizations to the training method and many more improvements made it the fastest and most accurate real-time model for object detection. All the above together with the facts that it is an one stage detector with state-of-the-art performance and that succeeds better accuracy than Mask R-CNN to MS COCO dataset [35], made us to utilize it and test it's performance to BDD100K and Volvo's dataset.

3.1 Tools

Object detection is a complex technique that is hard to be implemented without using existing libraries and frameworks. Below are some of the tools and algorithms that are used in this thesis.

Tensorflow

TensorFlow is an open source software framework for machine learning. It has a comprehensive ecosystem of tools, libraries and community resources for machine learning. TensorFlow can be used in a variety of devices, such as mobile devices, or

CPU/GPU clusters. It has been developed within Google, written in C++ and it can be accessed using API like Python, C and C++[1].

Keras

Keras is a deep learning library that supports convolutional and recurrent networks. It is written in Python and it can be run on top of TensorFlow, CNTK, or Theano. It can run both on CPU and GPU[2].

CUDA

CUDA [20] is a parallel computing platform created by Nvidia. CUDA enables users to run parts of their code on the GPU and speed up execution. The speed-up is achieved through exploiting the GPU for specific operations such as matrix multiplication, that GPUs can perform more effectively than CPUs. Matrix multiplication is used extensively when performing both forward and back propagation through a neural network, meaning that CUDA enables significant speed-up when training deep neural network.

3.2 Datasets

Training a deep neural network requires a large amount of data that should be relevant to the case of study. Collecting and annotating this type of data takes a significant amount of time. For this reason it is essential to use transfer learning (see Section 2.7).

In this project it is used the Berkeley DeepDrive dataset (BDD100K) one of the most popular public autonomous driving datasets available for research purposes. Additionally, Volvo's data, that is collected from the front camera of the reference box of one of Volvo's vehicles, is also used for training and testing. We decided to use BDD100K dataset because it is closely related to Volvo's data with very similar categories and scenery. Therefore, BDD100K provides a good pre-trained network so that we can utilize and train on Volvo's data.

3.2.1 Berkeley DeepDrive dataset

Berkeley DeepDrive dataset[3] [21] is a large-scale driving video dataset with extensive annotations for heterogeneous tasks. For this thesis it is used part from the image dataset that consists approximately 100000 images which are annotated with 2D bounding boxes. In particular it uses a set of 69863 images for training the network and a set of 10000 images for validating/testing its performance. Ten object categories are available: bus, traffic light, traffic sign, person, bike, truck, motor, car,

[1] `https://www.tensorflow.org/`
[2] `https://keras.io/`
[3] `The dataset was downloaded from the web page: https://bdd-data.berkeley.edu/`

train, and rider. Figure 3.1 shows a long-tail distribution histogram that represents the number of instances of each category of BDD100K dataset.

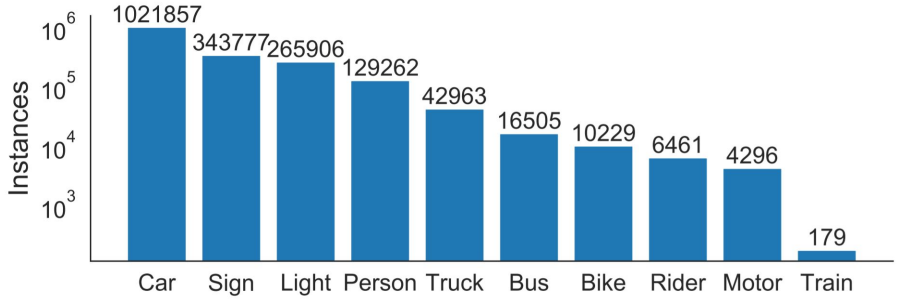


Figure 3.1: Number of instances in each category of BDD100K dataset [21].

The images are collected from many cities and regions in US (New York, San Francisco Bay Area, and other regions). They contain large portions of extreme weather conditions, such as snow and rain. They also include a diverse number of different scenes across the world and contain approximately an equal number of day-time and night-time cases. The dimensions of all images are 1280/720 pixels.

3.2.2 Volvo's Data

In this thesis it is used a set of approximately 1200 images collected from the camera on the reference box of one of Volvo's vehicles. In particular it is used a set of 914 images for training the network and a set of 228 images for validating/testing its performance. Annotation is been held with the help of one of Volvo's annotation tools for the needs of this master thesis. The object categories are five: car, big vehicle (including buses and trucks), pedestrian, motorcycle (with rider) and bicycle (with rider). They are all dynamic objects. The instances of each category is shown in Figure 3.2. The images are captured in different cities in Europe, day and night, with different weather conditions. The dimensions of all images are 4096/2176 pixels.

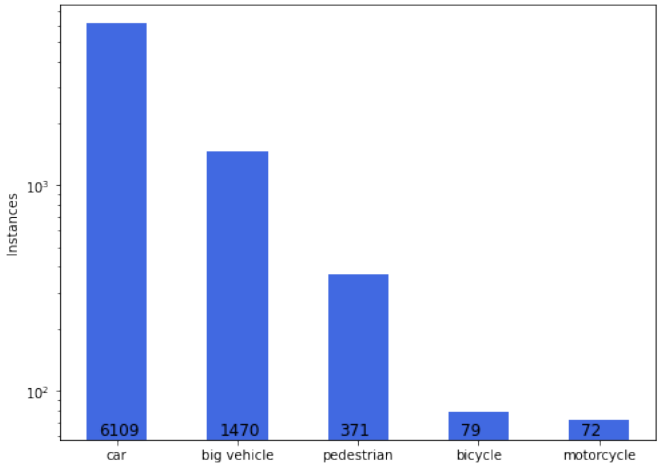


Figure 3.2: Number of instances in each category of Volvo's dataset.

3.3 Methods

3.3.1 Mask R-CNN

The Mask Region-based Convolutional Neural Network (Mask R-CNN) [5] is a two stage detector. Mask R-CNN uses the Faster R-CNN architecture and extends it by adding in parallel with the bounding box recognition branch, another branch for predicting the object's mask. (see Figure 3.3). The added branch is a fully convolutional network on top of a CNN based feature map, where the input is the CNN feature map and the output is a matrix with 1 if the pixel belongs to an object and 0 elsewhere, known as binary mask. Mask R-CNN model supports both object detection (bounding boxes) and object segmentation (masks). The datasets that are used for this project do not provide annotated masks, so we do not focus on the image segmentation abilities of the Mask R-CNN model.

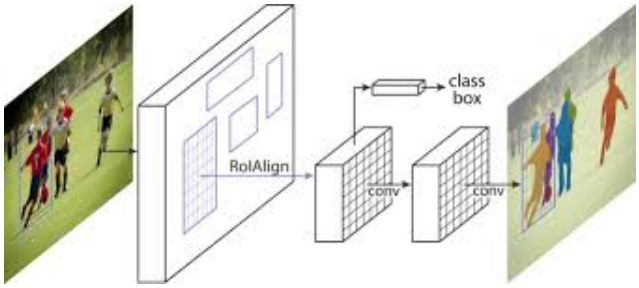


Figure 3.3: Mask R-CNN framework [5].

An implementation of the model from scratch would be time consuming so we used a third-party implementation build on top of the Keras deep learning framework. The name of it, is Mask R-CNN Project and it is developed by Matterport[4]. The mrcnn library uses Tensorflow for training the deep network and Python 3 programming language.
Our Mask R-CNN model has ResNet-101 [26] as a backbone and it is based on Feature Pyramid Network (FPN). The maximum detection instances in each image are 100.

3.3.1.1 Load and read the data

To load and read the data, Mask R-CNN library (mrcnn) requires to be created a dataset object. As a consequence, we created a class that extends the mrcnn.utils.Dataset class and we defined four functions to load the dataset, extract the boxes of each image, load the mask and load an image reference (path) respectively. To the function that loads the dataset, we specify the object classes, the paths for the images, the annotation files and how we split the training and validating set. To the function that extracts the boxes we specify how to gain the information of annotation that is included in the json files. To the function that loads the mask since we don't have masks, we just load the bounding boxes and return them as

[4] `https://github.com/matterport/Mask_RCNN.`

masks. The library then infers bounding boxes from our "masks" which are the same size. Finally the function that loads the image reference, returns the path of the image.

3.3.1.2 Train the model

We use transfer learning to train our model so that we can take advantage of pre-trained model on BDD100K dataset and then transfer it to Volvo's data. In particular, we used initially pre-trained MS COCO[5] weights to all layers apart from the output layers for the classification label, bounding boxes and masks. The output layers of the model, we trained them by using BDD100K dataset. Then we used transfer learning and trained this model more by using Volvo's data. For the training we used one Tesla T4 GPU with memory 14249MB. For training BDD100K dataset the learning rate was 0.001, the epochs were 5, the steps per epoch were 34932, the images per GPU were 2 and it took approximately 14,2 hours per epoch. For training Volvo's dataset, on the other hand, the learning rate was 0.001, the epochs were 8, the steps per epoch were 457 and the images per GPU were 2.

Loss Function

The loss function of Mask R-CNN is a combination of the classification loss, localization loss and segmentation mask loss:

$$\mathcal{L} = \mathcal{L}_{cls} + \mathcal{L}_{box} + \mathcal{L}_{mask} \tag{3.1}$$

where $\mathcal{L}_{cls}$ and $\mathcal{L}_{box}$ are same as in Faster R-CNN:

$$\mathcal{L}_{cls} = \frac{1}{N_{cls}} \sum_i \mathcal{L}_{cls}(p_i, p_i^*) = \frac{1}{N_{cls}} \sum_i (-p_i^* log p_i - (1 - p_i^*) log(1 - p_i)) \tag{3.2}$$

and

$$\mathcal{L}_{box} = \frac{\lambda}{N_{box}} \sum_i p_i^* \cdot L_1^{smooth}(t_i - t_i^*) \tag{3.3}$$

$\mathcal{L}_{mask}$ is the average binary cross-entropy loss. It only includes the k-th mask if the region is associated with the ground truth class k.

$$\mathcal{L}_{mask} = -\frac{1}{m^2} \sum\nolimits_{1 \le i,j \le m} [y_{ij} \log \hat{y}_{ij}^k + (1 - y_{ij}) \log(1 - \hat{y}_{ij}^k)] \tag{3.4}$$

where y_{ij} is the label of a cell (i, j) in the true mask for the region of size m x m; $\hat{y}_{ij}^k$ is the predicted value of the same cell in the mask learned for the ground-truth class k.
L_1^{smooth} is the smooth L1 loss.
p_i Predicted probability of anchor i being an object.
p_i^* Ground truth label (binary) of whether anchor i is an object.
t_i Predicted four parameterized coordinates.
t_i^* Ground truth coordinates.

[5] http://cocodataset.org/#home

N_{cls} Normalization term, set to be mini-batch size ($\sim$ 256).
N_{box} Normalization term, set to the number of anchor locations ($\sim$ 2400) in the paper Faster R-CNN [4].
λ A balancing parameter, set to be $\sim$ 10 in the paper Faster R-CNN [4] (so that both $\mathcal{L}_{cls}$ and $\mathcal{L}_{box}$ terms are roughly equally weighted).

3.3.1.3 Evaluation of the model

The performance of our object detection model was evaluated using the mean average precision, or mAP. It was also calculated the mAP of each category and the confusion matrix on Volvo's dataset.

3.3.2 YOLOv4

According to *YOLOv4: Optimal Speed and Accuracy of Object Detection* [25] an object detector is usually composed of a backbone which is pre-trained on ImageNet [15] and a head which predicts classes and bounding boxes of objects. The head is either one-stage or two-stage detector (Section 2.4). Recent object detectors usually insert some layers between backbone and head. In YOLOv4 [25] these layers are called neck and they are used to collect feature maps from different stages with several bottom-up and top-down paths. All the above can be seen schematically in Figure 3.4. Our implementation of YOLOv4 uses as a backbone CSPDARKNET53 [36], as a neck Path-Aggregation Network (PANet) [37] and as a head YOLOv3 [9].

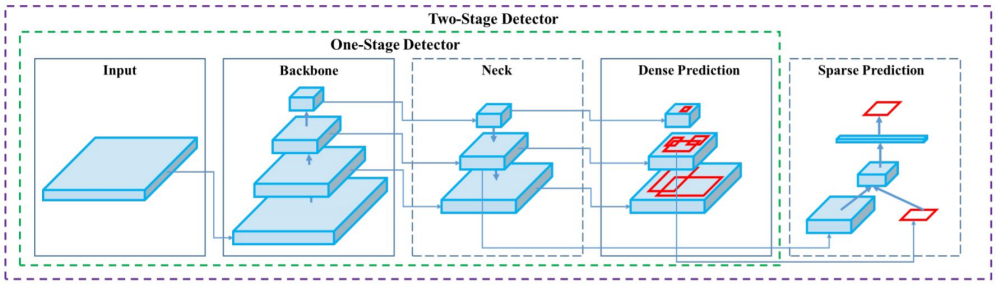


Figure 3.4: Architecture of recent object detectors [25].

YOLOv4 utilizes the CSP connections with the Darknet-53 that was used in YOLOv3 as the backbone in feature extraction (CSPDARKNET53). The letters CSP stands for Cross-Stage-Partial connections. These connections separate the input feature maps into two parts, one that goes through a block of convolutions, and one that does not. Then, the results are aggregated. The network that is used as a neck is a modified version of the PANet (Path Aggregation Network) [37]. The idea is to aggregate information to get higher accuracy.

Apart from YOLO's architecture that is liable for its good performance, there are also some optimizations that are called in YOLOv4 bag of freebies and bag of specials.

Bag of freebies are the optimizations to the training method that induce better accuracy without increasing the inference cost. For instance it is used data augmentation

which increases the variability of the input images, so that the trained model has higher robustness to the various images. The bag of freebies that are used to the backbone and the detector are shown in Table 3.1.

Location	Bag of freebies	Description
Backbone	CutMix [38]	Data Augmentation
	Mosaic	Data Augmentation
	DropBlock [39]	Regularization method
	Class label smoothing	Regularization Technique
Detector	CIoU-loss [40]	Bounding box regression loss
	CmBN	Normalization of the network activations by their mean and variance
	DropBlock [39]	Regularization method
	Mosaic	Data Augmentation
	Self-Adversarial Training (SAT)	Data Augmentation
	Eliminate Grid Sensitivity	Bounding box computation improvement
	multiple anchors for a single ground truth	threshold to assign a box as object or background: IoU(truth, anchor) > IoU_threshold
	Cosine annealing scheduler [41]	Learning rate adjustment
	Optimal hyper-parameters	Hyper-parameter selection using genetic algorithms
	Random training shapes	Automatic increase of mini-batch size during small resolution training by using Random training shapes

Table 3.1: Bag of freebies used in backbone and detector of YOLOv4

On the other hand, bag of specials are called the set of modules that improve the accuracy of object detection significantly with only increasing a little the inference cost. The bag of specials that are used to the backbone and the detector of YOLOv4 are shown in Table 3.2.
The improvements that were introduced in YOLOv4 [25] are the following. Mosaic is a new data augmentation strategy that combines four images into one for training instead of two that are used in CutMix [38]. This allows detection of objects outside their normal context and it significantly reduces the need for a large mini-batch size. On the other hand Class Label Smoothing is used for mitigating overfitting by adjusting the target upper bound of the prediction to a lower value to use this value in calculating the loss. This is useful so that it is avoided to memorize the data instead of learn it.
Moreover, Cross mini-Batch Normalization (CmBN) was also introduced in YOLOv4 [25]. It is a modified version of Cross-Iteration Batch Normalization (CBN) [42] that collects statistics only between mini-batches within a single batch, instead of collecting statistics inside a single mini-batch.

Location	Bag of specials	Description
Backbone	Mish activation [43]	Activation
	Cross-Stage Partial connections (CSP) [36]	Skip-connections
	Multi-input Weighted Residual Connections (MiWRC)	Skip-connections
Detector	Mish activation [43]	Activation
	SPP-block [12]	Additional blocks
	SAM-block [45]	Additional blocks
	PAN [37]	Path-aggregation blocks
	DIoU-NMS [40]	Bounding box regression loss

Table 3.2: Bag of specials used in backbone and detector of YOLOv4

Self-Adversarial Training (SAT) is another data augmentation technique that was introduced in YOLOv4 [25]. It works in two forward backward stages. In the first stage the model changes the image such that it can degrade the detector performance the most instead of changing the network weights as it is usually done in the backpropagation. In the second stage, the model is trained to detect an object on this modified image. This technique helps to generalize the model and to reduce overfitting.

Another optimization that was applied in YOLOv4 is the eliminate grid sensitivity where it is used a factor in the computation of the bounding box so that the effect of grid on which the object is undetectable can be eliminated.

Finally Multi-input Weighted Residual Connections (MiWRC) was introduced to the bi-directional feature pyramid network (BiFPN) [44] and was modified a bit to YOLOv4 [25] to make it suitable for efficient training and detection. MiWRC is proposed to execute scale-wise level re-weighting, and then add feature maps of different scales.

3.3.2.1 Train the model

An implementation of the model from scratch would be time consuming so we used the source code of YOLOv4 which is on the open source neural network framework called Darknet[6]. It is written in the C and Python programming languages and uses CUDA technology.

We use transfer learning to train our model. MS COCO pre-trained weights of the network were available, so we used them and then we train it by using BDD100K dataset. Finally we train it more by using Volvo's data. For the training we used one Tesla T4 GPU with memory 14249MB. For training BDD100K dataset the learning rate was 0.001, the epochs were 62, the batch size was 64, the mini batch size was 32, the images per GPU were 2 and it took approximately 2,5 hours per epoch. For

[6] `https://github.com/AlexeyAB/darknet`

training Volvo's dataset , on the other hand, the learning rate was 0.001, the epochs were 756, the batch size was 64, the mini batch size was 16 and the images per GPU were 4.

Loss Function

YOLO's multi-part loss function (localization, classification, confidence loss) that was introduced to [7] is presented below:

$$\begin{aligned} &\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2] \\ &+\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}\left[\left(\sqrt{w_i}-\sqrt{\hat{w}_i}\right)^2+\left(\sqrt{h_i}-\sqrt{\hat{h}_i}\right)^2\right] \\ &+\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}\left(C_i-\hat{C}_i\right)^2+\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{noobj}\left(C_i-\hat{C}_i\right)^2 \\ &+\sum_{i=0}^{S^2}1_{i}^{obj}\sum_{c\in classes}(p_i(c)-\hat{p}_i(c))^2 \end{aligned} \tag{3.5}$$

where 1_i^{obj} indicates if the object appears in grid cell i.
1_{ij}^{obj} indicates that the jth bounding box predictor in grid cell i is responsible for that prediction. 1_{ij}^{obj} is 1 if there is object in the cell and 0 if there is not.
1_{ij}^{noobj} is inverse of 1_{ij}^{obj}, where it is 1 if there is no object in the cell and 0 if there is.
λ_{coord} is a parameter that increases the weight for the loss in the boundary box coordinates.
λ_{noobj} is a parameter that weights down the loss when detecting background.
x_i, y_i is the location of the centroid of the anchor box.
S is the dimensions of the grid (SxS) that the input images is divided.
B the number of bounding boxes that are predicted for each grid cell.
w_i, h_i is the width and height of the anchor box.
$\hat{C}_i$ is the confidence score of the box j in cell i.
$\hat{p}_i(c)$ class probability for class c in cell i.
As we can see, w_i, h_i are under square root. The square root is present to penalize the smaller bounding boxes as we need to adjust them more since the small deviations in large boxes matter less than in small boxes.

One of the improvements in loss function that was added in YOLOv4 [25] is CIoU-loss [40]. CIoU-loss increases the overlapping area of the ground truth box and the predicted box, minimizes their central point distance, and maintains the consistency of the boxes' aspect ratio. The mathematical formula is the below:

$$\mathcal{L}_{CIoU}=1-IoU+\frac{\rho^2(\mathbf{b},\mathbf{b}^{gt})}{c^2}+\alpha\upsilon \tag{3.6}$$

Where ρ is the Euclidean distance, $\mathbf{b}$ and $\mathbf{b}^{gt}$ are the central points of the predicted and target box respectively, c is the diagonal length of the smallest enclos-

ing box covering the two boxes, υ measures the consistency of aspect ratio $\upsilon = \frac{4}{\pi^2}(arctan\frac{w^{gt}}{h^{gt}} - arctan\frac{w}{h})^2$ and α is the trade-off parameter defined as $\alpha = \frac{\upsilon}{(1-IoU)+\upsilon}$.

4

Results

4.1 Mask R-CNN Results

4.1.1 After training with BDD100K dataset

Since we had the pre-trained weights of MS COCO dataset available, we firstly calculated the mAP of the test set of 10000 images of BDD100K dataset to have it as a baseline. We have to take into consideration that the number of categories that MS COCO dataset contains is 80 and BDD100K contains only 10 which are not all of them included to COCO dataset. MS COCO dataset does not have *rider* and *traffic sign* categories that BDD100K dataset has. In particular we assumed that the BDD100K's category *rider* corresponds to the COCO's category *person*. Also COCO dataset doesn't detect all the *traffic signs*, but only the *stop sign*. As a result not all of the *traffic signs* were detected by this model. Consequently the result of the mAP calculation was $mAP_{test\_set} = 0.215$ or 21.5%.

As it was described in Section 3.3.1.2, we trained the output layers of the Mask R-CNN model with BDD100K dataset for 5 epochs. We chose the weights of epoch 5 because after measuring the mAP using the weights of the first 13 epochs, we saw that the mAP was decreasing after epoch 5 and since one epoch took approximately 14.2 hours, we decided not to continue training more.

We calculated the mAP of the train set (69863 images) and the mAP of the test set (10000 images) and we found that they are: $mAP_{train\_set} = 0.457$ and $mAP_{test\_set} = 0.456$. Compared to the baseline mAP, the performance of model was improved after training it to the BDD100K dataset and this was something that was expected. Table 4.1 presents the mAP of Mask R-CNN model in total and for each category after training it with BDD100K dataset. These results are from implementing the model to the BDD100K test set which has 10000 images. From the results on the Table 4.1, we can see that cars and humans have higher mAP values than the other categories. The lower mAP is for the category *train*, *motorcycle* and *bike*. We have to take into account that all classes impact the overall mAP.

Figure 4.1 presents a visualized example an image in the BDD100K test set where the ground truth boxes on the left image, and the predicted box on the right image. As we can see there are some false positive boxes where they predict wrongly an object instead of background but also there are many true positives.

mAP	0.456
mAP_{car}	0.553
mAP_{person}	0.359
$mAP_{bicycle}$	0.162
mAP_{truck}	0.258
mAP_{bus}	0.216
$mAP_{motorcycle}$	0.153
mAP_{rider}	0.484
$mAP_{traffic\_sign}$	0.377
$mAP_{traffic\_light}$	0.328
mAP_{train}	0.0

Table 4.1: Evaluation of Mask R-CNN model trained to BDD100K dataset.

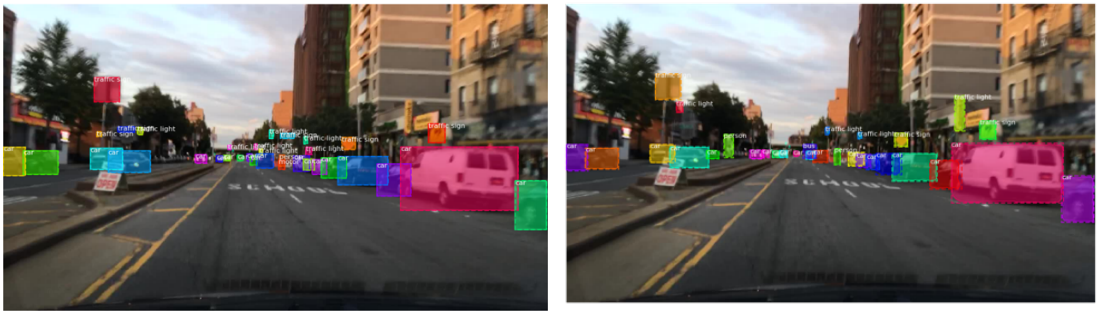


Figure 4.1: Visualized results of Mask R-CNN model: On the left, is the actual annotated boxes and on the right, is the predicted objects from Mask R-CNN model.

4.1.2 After training with Volvo's data

We used the previous model that was trained to BDD100K dataset to first calculate the mAP on Volvo's test set to have it as a baseline and then train it further with Volvo's train set and evaluate it so that we can compare.

Since BDD100K dataset has ten categories and Volvo's dataset has only 5 we had to make some assumption to calculate the accuracy of Volvo's test set. We assumed that the categories of BDD100K dataset, *truck* and *bus* belong to the same category of Volvo's dataset named *big vehicle*. Also in BDD100K dataset there is a separate category for the rider of the bicycle and the motorcycle named *rider* but in Volvo's dataset the categories *bicycle* and *motorcycle* include the rider in the bounding box. So our assumption was that the bounding box of the bicycle with or without a rider has not a big difference in terms of the area, the same was assumed for the motorcycle. So we ignored the rider and assumed that the *bike* and *motor* from BDD100K is the same with the *bicycle* and *motorcycle* from Volvo's dataset. Of course there may be the case that the model trained to BDD100K dataset detects a bicycle parked on the side of the road having no rider, but this object is not annotated in Volvo's dataset since we are only interested for dynamic objects. Consequently, the mAP of Volvo's test set using the previous model trained to BDD100K dataset was $mAP = 0.543$ or 54.3%.

As it was described in Section 3.3.1.2 we trained the output layers of the Mask R-CNN model with Volvo's train set for 8 epochs. We chose the weights of epoch 8 because after measuring the mAP using the weights of the first 12 epochs, we saw that the mAP was decreasing after epoch 8 so we stopped the training.

We calculated the mAP of the train set (914 images) and the mAP of the test set (228 images) and we found that they are: $mAP_{train\_set} = 0.676$ and $mAP_{test\_set} = 0.631$. As we can see the performance of the model to Volvo's test set was improved after training it to Volvo's dataset as it was expected. In the Table 4.2 it is presented the mAP of our model in total and for each category after training it with Volvo's dataset tested to Volvo's test set (228 images).

mAP	0.631
mAP_{car}	0.682
$mAP_{pedestrian}$	0.549
$mAP_{big\_vehicle}$	0.591
$mAP_{bicycle}$	0.348
$mAP_{motorcycle}$	0.75

Table 4.2: Evaluation of Mask R-CNN model after trained to Volvo's dataset.

The confusion matrix of Volvo's test set is also presented in Table 4.3. The diagonal of the matrix (yellow cells) shows the correct classification results. The right column (green cells) shows the actual objects that where not detected, in other words they were detected as a background. So this column shows the False Negative cases. Finally the last row (red cells) shows the wrong detections where there was not an actual object there (ghost object). These are the False Positive cases. As we can see from this matrix, 883 cars were detected correctly as cars and 3 were detected wrongly as big vehicles, 27 pedestrians were detected correctly as pedestrians and 1 was detected wrongly as bicycle, 198 big vehicles were detected correctly as big vehicles and 21 were detected wrongly as cars, 5 bicycles were detected correctly as bicycles, 1 was detected wrongly as pedestrian and 1 was detected wrong as motorcycle. Finally 13 motorcycles were detected correctly as motorcycles and 1 was detected wrongly as car.

		Predicted Classes					
		car	pedestrian	big vehicle	bicycle	motorcycle	background
Actual Classes	car	883	0	3	0	0	391
	pedestrian	0	27	0	1	0	36
	big vehicle	21	0	198	0	0	95
	bicycle	0	1	0	5	1	11
	motorcycle	1	0	0	0	13	4
	background	510	21	95	1	2	0

Table 4.3: Confusion Matrix of Mask R-CNN model on Volvo's dataset

Figure 4.2 presents an example from the Volvo's dataset with the ground truth boxes on the left, and the predicted boxes from Mask R-CNN model on the right. As we

can see in general the predictions are accurate but there is a detected big vehicle instead of a background.

Figure 4.2: Visualized results of Mask R-CNN to Volvo's dataset: On the left, is the actual annotated image and on the right, is the predicted objects from our model.

4.2 YOLOv4 Results

4.2.1 After training with BDD100K dataset

Utilizing the available pre-trained weights of MS COCO dataset for YOLOv4 model, we calculated the mAP of the test set of 10000 images of BDD100K dataset to have it as a baseline as we did to the previous model. As it was mentioned, the number of categories of MS COCO dataset differs from BDD100K dataset. For this reason we did the same assumptions, we assumed that the category *rider* that BDD100K has is the same as the category *person* that COCO has and the category *traffic signs* from BDD100K is the same as *stop sign* from COCO. Thus, the result of the mAP calculation before training was $mAP_{test\_set} = 0.346$ or 34.6%.

As it was described in Section 3.3.2.1, we trained YOLOv4 model with BDD100K dataset for 62 epochs. Specifically, we trained it for 72 epochs, but the best weights with the best mAP were in epoch 62. We calculated the mAP of the test set (10000 images) and we found that it is: $mAP_{test\_set} = 0.693$. The performance of our model was improved after training it to the BDD100K dataset as it was expected. In the Table 4.4 it is presented the mAP of our model in total and for each category after training it with BDD100K dataset tested to BDD100K test set (10000 images).

Table 4.4 shows that all categories have relatively high mAP except from *motorcycle* and *bicycle*. All values of mAP are higher than the corresponding ones from Mask R-CNN (Table 4.1). A more detailed comparison will be held in Section 4.3.

In Figure 4.3 we presented an example image of BDD100K dataset with the ground truth bounding boxes to the left figure, and the predicted ones to the right figure from YOLOv4 model. As we can see there is a false positive box where it predicts wrongly a car instead of background but also there are some cars far away that are not detected.

mAP	0.693
mAP_{car}	0.734
mAP_{person}	0.492
$mAP_{bicycle}$	0.416
mAP_{truck}	0.552
mAP_{bus}	0.517
$mAP_{motorcycle}$	0.35
mAP_{rider}	0.579
$mAP_{traffic\_sign}$	0.666
$mAP_{traffic\_light}$	0.583
mAP_{train}	0.0

Table 4.4: Evaluation of YOLOv4 model trained to BDD100K dataset.

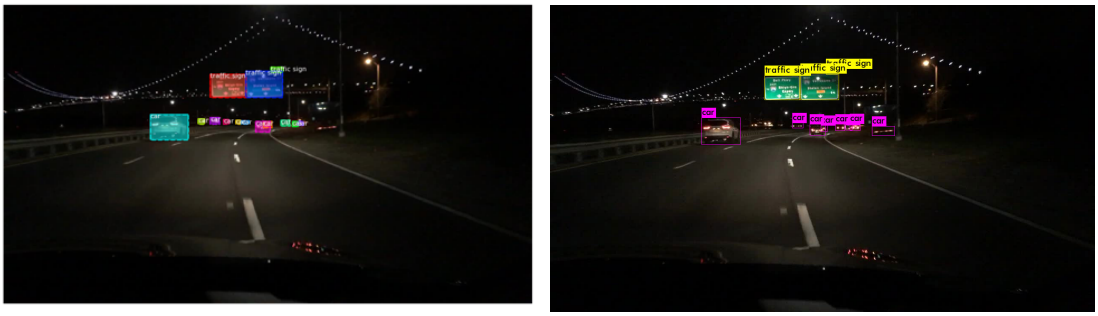


Figure 4.3: Visualized results of YOLOv4 model: On the left, is the actual annotated boxes and on the right, is the predicted objects from YOLOv4 model.

4.2.2 After training with Volvo's data

Before we start further training of the model with Volvo's dataset, we calculated, firstly, the mAP on Volvo's test set using the weights after training with BDD100K, to have it as a baseline.

We make the same assumptions for the categories of BDD100K and Volvo's dataset as we did to the Mask R-CNN model. Particularly we assumed that the *truck* and *bus* categories of BDD100K dataset correspond to the *big vehicle* category of Volvo's dataset. Also we assumed that the bounding box of the bicycle or the motorcycle with or without a rider has not a big difference in area. So we ignored the rider and assumed that the *bike* and *motor* from BDD100K is the same with the *bicycle* and *motorcycle* from Volvo's dataset respectively. Thus, the mAP of the YOLOv4 model trained to BDD100K dataset applied to Volvo's test set, is $mAP = 0.535$ or 53.5%.

Subsequently we further trained the YOLOv4 model with Volvo's train set (914 images) for 756 epochs. Specifically, we trained it for 980 epochs, but the best weights (best mAP) were in epoch 756. We calculated the mAP of the test set (228 images) and we found that it is: $mAP_{test\_set} = 0.755$. As it is obvious the performance of our model to Volvo's test set was improved after training it to Volvo's dataset as it was expected. In the Table 4.5 it is presented the mAP of YOLOv4

model in total and for each category after training it with Volvo's dataset tested to Volvo's test set. As we can see *motorcycle* category has the best mAP and *pedestrian* category has the lowest mAP.

mAP	0.755
mAP_{car}	0.787
$mAP_{pedestrian}$	0.66
$mAP_{big\_vehicle}$	0.742
$mAP_{bicycle}$	0.795
$mAP_{motorcycle}$	0.875

Table 4.5: Evaluation of YOLOv4 model after trained to Volvo's dataset.

		Predicted Classes					
		car	pedestrian	big vehicle	bicycle	motorcycle	background
Actual Classes	car	979	0	10	0	0	288
	pedestrian	0	39	0	2	0	23
	big vehicle	6	0	234	0	0	74
	bicycle	0	1	0	14	0	3
	motorcycle	0	0	0	0	15	3
	background	273	22	64	2	3	0

Table 4.6: Confusion Matrix of YOLOv4 model on Volvo's dataset

Figure 4.4: Visualized results of YOLOv4 to Volvo's dataset: On the left, is the actual annotated image and on the right, is the predicted objects from YOLOv4 model

It is also calculated the confusion matrix of Volvo's test set of YOLOv4 model and is presented in Table 4.6. As we described before, the diagonal of the matrix (yellow cells) shows the correct detections, the right column (green cells) shows the False Negative cases and the last row (red cells) shows the False Positive cases. As we can see from this matrix, 979 cars were detected correctly as cars and 10 were detected wrongly as big vehicles, 39 pedestrians were detected correctly as pedestrians and 2 were detected wrongly as bicycle, 234 big vehicles were detected correctly as big vehicles and 6 were detected wrongly as cars, 14 bicycles were detected correctly as

bicycles and 1 was detected wrongly as pedestrian and finally 15 motorcycles were detected correctly as motorcycles and 3 were not detected at all (background).

In Figure 4.4 we presented an example of Volvo's dataset with the ground truth bounding boxes on the left and the predicted ones of YOLOv4 model on the right. As we can see in general the predictions are accurate except from a wrongly detected hidden car instead of a background.

4.3 Comparison of Mask R-CNN and YOLOv4 models

In this Section are presented some tables and plots to have a clearer picture of the two models so that the comparison is easier.

			Models	
			Mask R-CNN	YOLOv4
Test Set	BDD100K	$mAP_{before\ train(COCO\ weights)}$	0.215	0.346
		$mAP_{after\ train(BDD100K\ weights)}$	0.456	0.693
	Volvo	$mAP_{before\ train(BDD100K\ weights)}$	0.543	0.535
		$mAP_{after\ train(VOLVO\ weights)}$	0.631	0.755

Table 4.7: Total mAP of Mask R-CNN and YOLOv4 models before and after training them to BDD100K and Volvo's dataset tested to BDD100K and Volvo's test sets.

In Table 4.7 we presented the total mAP of BDD100K and Volvo's test sets calculated before and after training them with these two datasets for both Mask R-CNN and YOLOv4 models. In all cases YOLOv4 has higher mAP than Mask R-CNN apart from the case of Volvo's test set that the mAP is calculated before training the model with Volvo's data. Over there YOLOv4 has a bit lower mAP (0.535) than Mask R-CNN (0.543). Also, in all cases the performance of each model to each test set is better after training it with the corresponding dataset.

In Figure 4.5 we presented the mAP of each category of BDD100K dataset for Mask R-CNN and YOLOv4 models. As we can see in all cases YOLOv4 outperforms Mask R-CNN. *Car* is the category that is detected better in both models and *train* is the worse detected category.
In Figure 4.6 it is presented the mAP of each category of Volvo's dataset for Mask R-CNN and YOLOv4 models. As it was observed before, in all cases YOLOv4 outperforms Mask R-CNN but here the difference between the two models in each category is lower apart from the category *bicycle* where YOLOv4 model performs more than twice as good as Mask R-CNN. *Motorcycle* is the category that is detected better in both models and *bicycle* is the worse detected category for Mask R-CNN and *pedestrian* is the worse detected category for YOLOv4.
Moreover it is quoted Table 4.8 and Table 4.9 with the intention to compare the results of the confusion matrices of the two models in Tables 4.3 and 4.6. Table 4.8 shows the percentages of the False Positive cases on Volvo's test set for Mask

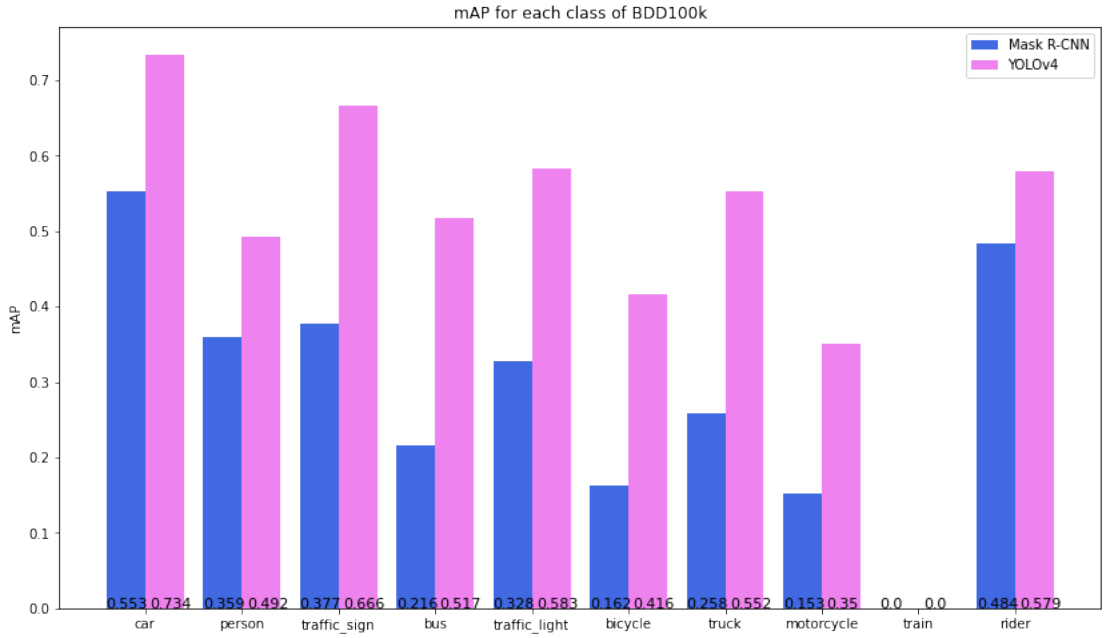


Figure 4.5: Histogram of mAP for each class of BDD100K dataset for both Mask R-CNN and YOLOv4 models.

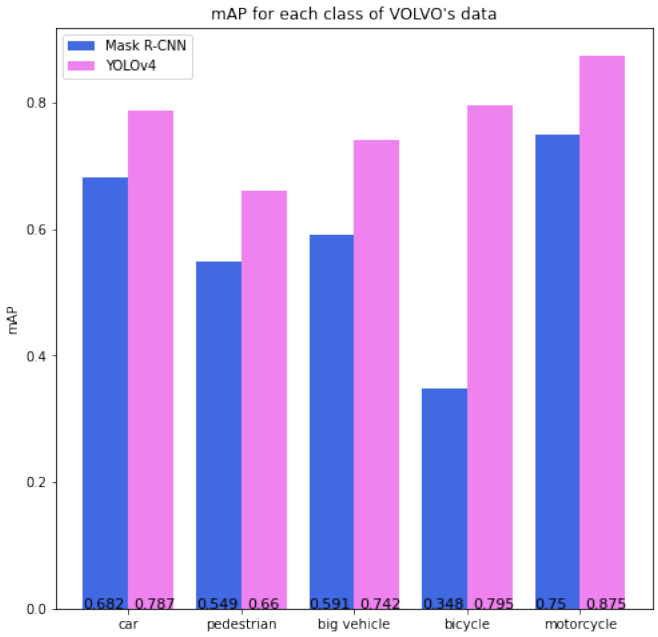


Figure 4.6: Histogram of mAP for each class of Volvo's dataset for both Mask R-CNN and YOLOv4 models.

R-CNN and YOLOv4 models. It is reminded that False Positive are the cases where there are detected objects that are not actual there or the IoU is less than the threshold. In all cases YOLOv4 has lower percentages than Mask R-CNN except from *motorcycle* where is a bit higher. The low values of YOLOv4 shows that it has less wrong detections than Mask R-CNN.

False Positive		
	Mask R-CNN	YOLOv4
Car	36.04%	**21.7%**
Pedestrian	42.86%	**35.48%**
Big vehicle	32.09%	**20.78%**
Bicycle	14.28%	**11.11%**
Motorcycle	**12.5%**	16.67%
TOTAL	35.28%	**21.87%**

Table 4.8: Percentage of False Positive cases for Mask R-CNN and YOLOv4 on Volvo's test set.

Table 4.9 on the other hand shows the percentages of the False Negative cases where an actual object is not detected (failure cases). YOLOv4 has lower values to all categories. This means that YOLOv4 detects more objects than Mask R-CNN does. It is also quoted Figure 4.7 with False Negative cases (failure cases) to have a graphical picture of what is shown in Table 4.9. From this graph we can see that *pedestrians* and *bicycles* are the categories that Mask R-CNN mostly misses to detect.

False Negative		
	Mask R-CNN	YOLOv4
Car	30.06%	**22.55%**
Pedestrian	56.25%	**35.93%**
Big vehicle	30.25%	**23.57%**
Bicycle	61.11%	**16.66%**
Motorcycle	22.22%	**16.66%**
TOTAL	31.76%	**23.12%**

Table 4.9: Percentage of False Negative cases for Mask R-CNN and YOLOv4 on Volvo's test set.

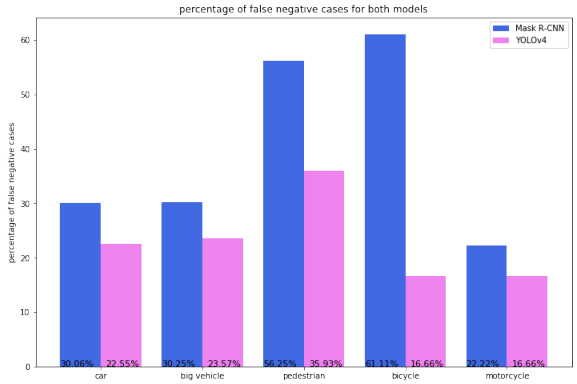


Figure 4.7: Histogram of percentage of False Negative cases for Mask R-CNN and YOLOv4 on Volvo's test set.

Finally we measured the inference time of both models. The inference time of Mask R-CNN is approximately 3.3482 sec or 3348.2 milli-seconds and the inference time of YOLOv4 was 34.582 milli-seconds. Mask R-CNN is almost 100 times slower than YOLOv4 in Volvo's dataset.
Figure 4.8 and Figure 4.9 are two images from Volvo's dataset. They present the actual image, the annotation and the prediction of Mask R-CNN model and YOLOv4 model. In both cases we can see that the objects that are near the camera are detected correctly, mistakes happen when the objects are far from the camera.

a) Actual image

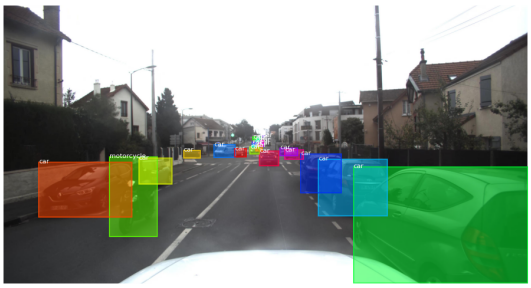
b) Annotated image

c) Mask R-CNN detection

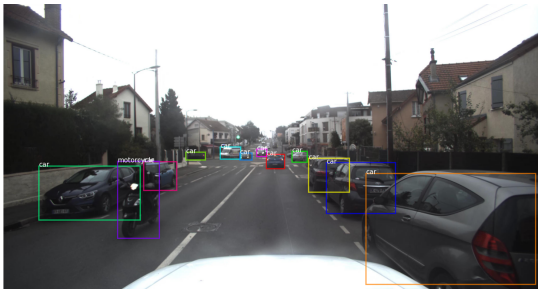
d) YOLOv4 detection

Figure 4.8: Visualized results of an images of Volvo's test set.

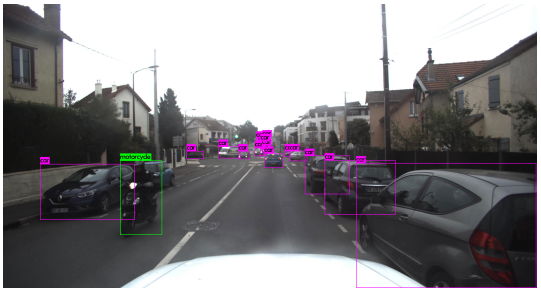
a) Actual image



b) Annotated image



c) Mask R-CNN detection



d) YOLOv4 detection

Figure 4.9: Visualized results of an images of Volvo's test set.

5
Discussion

5.1 Annotation of Datasets

The values of mAP in Table 4.1, Table 4.4 and graphically results in Figure 4.5 for the category *train* of BDD100K dataset, made us explore what is going on with the annotation of some images of BDD100K dataset. We realized that there are some errors, where trains either are missed and are not annotated, or there are some objects that belong to different category, wrongly annotated as trains. We are not that much interested the final model to detect trains but what made us sceptical is if these small mistakes are decreasing the performance of our models.
On the other hand before we start annotating Volvo's images, we designated to annotate the objects that are not hidden from other objects in a way to be hard to recognize. For example in Figure 5.1 it is shown the original image and the annotation made for this image. The yellow box at the right bottom side of each image is a zoom in of the central box. As we can see in the picture there are four to five cars in a row where some of them are hidden by the others. We decided to annotate two of them, the ones that are obvious. In Figure 5.2 it is shown the predictions of Mask R-CNN model and YOLOv4 model. Mask R-CNN in this example detects almost all the cars including the hidden ones that were not annotated. From the other side YOLOv4 detects the annotated cars but also detects wrongly a pedestrian and misses one pedestrian that is far away. This observe made us understand a bit more the high values of False Positives in confusion matrices in Tables 4.3 and 4.6.

Figure 5.1: Annotation of an image of Volvo's dataset.

Finally we believe that it was problematic that we had different categories in BDD100K and Volvo's dataset and maybe if we changed from the beginning the categories of BDD100K, to be the same as Volvo's dataset and train from the beginning like this, maybe the models would perform better.

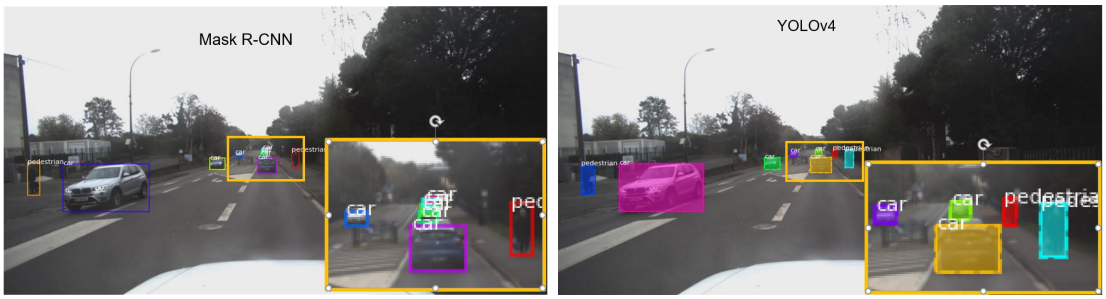


Figure 5.2: Predictions of an image of Volvo's dataset.

5.2 Semi-Supervised Learning

For the needs of this thesis it was annotated approximately 1200 images by placing a 2D bounding box to objects that belonged to one of the 5 categories of interest. This experience made us realize how difficult and time-consuming this procedure is. As a result we started searching for different methods where less annotated data are required and have the same and sometimes better results. So we ended up to the semi-supervised method for object detection. In particular we implemented a semi-supervised learning framework along with a data augmentation strategy for object detection that is described in [34] and it is called STAC. STAC is a simple and effective framework for object detection that claims to be more data efficient than the baseline models and brings significant gain in mAPs. We decided to use STAC because it is a state-of-the-art method with great results and simple to use.

STAC combines self training via pseudo label (see Section 2.5.2) of unlabeled images and consistency regularization based on strong data augmentations (see Section 2.5.1). The training is been held in two stages as it happens to the Noisy Student framework (see Section 2.5.3). In the first stage Faster R-CNN [4] model is trained to labeled data and then it is used to predict the classes and bounding boxes to the unlabeled images. Then, it is applied to each predicted box a confidence-based box filtering with high threshold value. The output pseudo labels are with high precision. In the second stage the model is trained with labeled and unlabeled data with the pseudo labels having strong data augmentations. The data augmentation strategy consists of global color transformation, global or box-level geometric transformations [46] and Cutout [47].

In our implementation we utilized the described framework and set BDD100K train set (69863 images) to be the labeled dataset and Volvo's train set (914) to be the unlabeled dataset. We resized Volvo's images so that they have the same size as the ones from BDD100K and we used only five of BDD100K's categories the ones that are in Volvo's dataset. Again we had to make the assumption that the bounding box of a bicycle or a motorcycle having rider and not having a rider has almost the same area and that *truck* and *bus* categories from BDD100K dataset are corresponding to the Volvo's category *big vehicle*.

The preliminary results from this attempt are shown in Table 5.1. In particular it is presented the measurements of mAP during the procedure of training STAC to BDD100K and Volvo's dataset. We calculated the mAP of the two test sets after

the first stage where only labeled data are used for training and after the last stage where labeled and unlabeled data are used and we have the final model. We have to take into consideration that the calculation of mAP in this case is calculated with a different function than in Mask R-CNN and YOLOv4 so the comparison with the mAP values of the two detectors is not valid.

	BDD100K test set	Volvo's test set
mAP after stage 1	0.588	0.602
mAP after stage 2	0.569	0.530

Table 5.1: Preliminary results of STAC. Calculation of mAP to both BDD100K and Volvo's dataset after stage 1 where the Faster RCNN model is trained to labelled data only and after stage 2 were the STAC model is completely trained to labeled and unlabeled data with pseudo-labels.

As we can see in Table 5.1 the mAP decreases after training with unlabeled data which was not our intention. We believe that this happened because our unlabeled dataset was very small compared with the ones in the [34]. Also the labeled and unlabeled sets in our case were from different datasets with different categories and size of images which made it more difficult to calculate good quality of pseudo-labels.

Due to the limit of time we leave to future work the test of efficacy of STAC to a bigger dataset of Volvo's unlabeled data, using as labeled data the ones we already have from Volvo. We believe that this method is very promising.

6

Conclusion and Future Work

In this thesis we presented an implementation of two deep learning object detection methods to two driving related datasets. In particular we utilized Mask R-CNN and YOLOv4 methods and BDD100K and Volvo's annotated datasets.

We have concluded that it is possible to apply the algorithms with high accuracy to the reference data of Volvo's vehicles. We succeeded to achieve better performance after training the two models with the two driving related datasets. We observed that YOLOv4 is better in terms of speed and mAP values than Mask R-CNN. So we can say that YOLOv4 is the best suited method for post-processing reference data from cameras.

As a future work we could have some improvements to our mAP if we train from the beginning the two models with the five classes of interest, and if we could double check the annotations of Volvo's data. Also, we leave as a future work the implementation of the Semi-supervised technique STAC with using more Volvo's unlabeled data. This technique is very promising as it claims to improve object detection model performance by using unlabeled data which is very useful since large-scale annotated data for our task is not available at the moment. STAC combines self-training and consistency regularization based on the strong data augmentations and improves the $AP^{0.5}$ of VOC07 dataset from 76.30 to 79.08 and demonstrates 2 $\times$ higher data efficiency on MS-COCO by achieving 24.38 mAP using only 5% labeled data than supervised baseline that marks 23.86% using 10% labeled data [34]. These results from the paper [34] shows that the implementation of STAC in our case can be very interesting. We can, additionally, use the calibration files that Volvo has to extract word coordinates and finally we can fuse the output of our method with LiDAR data to have a better reference system.

Bibliography

[1] Stephen E. Reutebuch, Hans-Erik Andersen, Robert J. McGaughey. (2005) Light Detection and Ranging (LIDAR): An Emerging Tool for Multiple Resource Inventory. Journal of Forestry.

[2] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik, UC Berkeley and ICSI. (2014) Rich feature hierarchies for accurate object detection and semantic segmentation. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[3] Girshick, R. (2015). Fast R-CNN. In Proceedings of the IEEE international conference on computer vision.

[4] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems.

[5] Piotr Dollár, Ross Girshick, Kaiming He, Georgia Gkioxari. (2018) Mask R-CNN. arXiv:1703.06870v3.

[6] Y. Li, K. He, J. Sun, J. Dai. (2016) R-fcn: Object detection via region-based fully convolutional networks. In NIPS.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. (2016) You only look once: Unified, real-time object detection. In CVPR.

[8] J. Redmon and A. Farhadi. (2016) Yolo9000: better, faster, stronger. arXiv:1612.08242.

[9] J. Redmon and A. Farhadi. (2018) YOLOv3: An incremental improvement. arXiv:1804.02767.

[10] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. (2016) Ssd: Single shot multibox detector. In ECCV.

[11] C. Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg. (2017) Dssd: Deconvolutional single shot detector. arXiv:1701.06659.

[12] K. He, X. Zhang, S. Ren, and J. Sun. (2015) Spatial pyramid pooling in deep convolutional networks for visual recognition. IEEE Trans. Pattern Anal. Mach. Intell.

[13] Z. Q. Zhao, P. Zheng, S. T. Xu, X. Wu. (2019) Object detection with deep learning: a review. IEEE Transactions on Neural Networks and Learning Systems.

[14] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders. (2013) Selective search for object recognition. Int. J. of Comput. Vision.

[15] A. Krizhevsky, I. Sutskever, and G. Hinton. (2012) ImageNet classification with deep convolutional neural networks. In NIPS

[16] M. Everingham, L. V. Gool, C. Williams, J. Winn, and A. Zisserman. (2008) The pascal visual object classes challenge 2007 (voc 2007) results results (2007).

[17] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar. (2018) Focal loss for dense object detection. IEEE transactions on pattern analysis and machine intelligence.

[18] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng and Rong Qu. (2019) A Survey of Deep Learning-based Object Detection. arXiv preprint arXiv:1907.09408v2.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. (2015) Deep Residual Learning for Image Recognition. arXiv:1512.03385v1.

[20] David Kirk et al. (2007) Nvidia cuda software and gpu parallel computing architecture. In ISMM, volume 7, pages 103–104.

[21] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, Trevor Darrell. (2020) BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning. arXiv:1805.04687v2.

[22] Bichen Wu, Forrest Iandola, Peter H. Jin, and Kurt Keutzer. (2016) Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. CoRR, vol. abs/1612.01051.

[23] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, Tian Xia. (2017) Multi-View 3D Object Detection Network for Autonomous Driving. In CVPR.

[24] Lin T.-Y., Dollar P., Girshick R., He K., Hariharan B., and Belongie S. (2017) Feature pyramid networks for object detection. In CVPR.

[25] Alexey Bochkovskiy, Chien-Yao Wang and Hong-Yuan Mark Liao. (2020) YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv:2004.10934v1.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. (2015) Deep Residual Learning for Image Recognition. arXiv:1512.03385v1.

[27] Kihyuk Sohn, Zizhao Zhang, Chun-Liang Li, Han Zhang, Chen-Yu Lee, Tomas Pfister. (2020) A Simple Semi-Supervised Learning Framework for Object Detection. arXiv:2005.04757.

[28] C. Cortes, V. Vapnik. (1995) Support vector machine. Machine Learning, vol. 20, no. 3, pp. 273–297.

[29] Q. Xie, E. Hovy, M. T. Luong, Q. V. Le. (2019) Self-training with Noisy Student improves ImageNet classification. arXiv:1911.04252.

[30] K. Sohn, D. Berthelot, C. L. Li, Z. Zhang, N. Carlini, E. D. Cubuk, A. Kurakin, H. Zhang, C. Raffel. (2020) Fixmatch: Simplifying semi-supervised learning with consistency and confidence. arXiv:2001.07685.

[31] Viktor Olsson, Wilhelm Tranheden, Juliano Pinto, Lennart Svensson. (2020) CLASSMIX: Segmantation-Based data augmentation for Semi-Supervised Learning. arXiv:2007.07936v1.

[32] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, Matti Pietikainen. (2019) Deep Learning for Generic Object Detection: A Survey. arXiv:1809.02165v4.

[33] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, Jieping Ye. (2019) Object Detection in 20 Years: A Survey. arXiv:1905.05055v2.

[34] Kihyuk Sohn, Zizhao Zhang, Chun-Liang Li, Han Zhang, Chen-Yu Lee, Tomas Pfister. (2020) A Simple Semi-Supervised Learning Framework for Object Detection. arXiv:2005.04757v1.

[35] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, Piotr Dollár. (2014) Microsoft COCO: Common Objects in Context. arXiv:1405.0312v3.

[36] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, I-Hau Yeh. (2020) CSPNet: A new backbone that can enhance learning capability of cnn. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshop (CVPR Workshop).

[37] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, Jiaya Jia. (2018) Path aggregation network for instance segmentation. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 8759–8768.

[38] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, Youngjoon Yoo. (2019) CutMix: Regularization strategy to train strong classifiers with localizable features. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), pages 6023–6032.

[39] Golnaz Ghiasi, Tsung-Yi Lin, Quoc V Le. (2018) DropBlock: A regularization method for convolutional networks. In Advances in Neural Information Processing Systems (NIPS), pages 10727–10737.

[40] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, Dongwei Ren. (2020) Distance-IoU Loss: Faster and better learning for bounding box regression. In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI).

[41] Ilya Loshchilov, Frank Hutter. (2016) SGDR: Stochastic gradient descent with warm restarts. arXiv:1608.03983.

[42] Zhuliang Yao, Yue Cao, Shuxin Zheng, Gao Huang, Stephen Lin. (2020) Cross-iteration batch normalization. arXiv:2002.05712.

[43] Diganta Misra. (2019) Mish: A self regularized nonmonotonic neural activation function. arXiv:1908.08681.

[44] Mingxing Tan, Ruoming Pang, Quoc V Le. (2020) EfficientDet: Scalable and efficient object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[45] Sanghyun Woo, Jongchan Park, Joon-Young Lee, In So Kweon. (2018) CBAM: Convolutional block attention module. In Proceedings of the European Conference on Computer Vision (ECCV), pages 3–19.

[46] Zoph, B., Cubuk, E.D., Ghiasi, G., Lin, T.Y., Shlens, J., Le, Q.V.. (2019) Learning data augmentation strategies for object detection. arXiv:1906.11172.

[47] DeVries, T., Taylor, G.W.. (2017) Improved regularization of convolutional neural networks with cutout. arXiv:1708.04552.

[48] Yosinski J, Clune J, Bengio Y, Lipson H. (2014) How transferable are features in deep neural networks? In Advances in Neural Information Processing Systems 27, NIPS Foundation.

[49] Lisa Torrey, Jude Shavlik. (2009) Transfer Learning. IGI Global.

[50] D. G. Lowe. (2004) Distinctive image features from scale-invariant keypoints. Int. J. of Comput. Vision, vol. 60, no. 2, pp. 91–110.

[51] N. Dalal, B. Triggs. (2005) Histograms of oriented gradients for human detection. in CVPR.

[52] R. Lienhart, J. Maydt. (2002) An extended set of haar-like features for rapid object detection. in ICIP.

[53] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, D. Ramanan. (2010) Object detection with discriminatively trained part-based models. IEEE Trans. Pattern Anal. Mach. Intell., vol. 32, pp. 1627–1645.

[54] Y. Freund, R. E. Schapire. (1997) A desicion-theoretic generalization of on-line learning and an application to boosting. J. of Comput. & Sys. Sci., vol. 13, no. 5, pp. 663–671.

[55] S. Ioffe, C. Szegedy. (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

[56] Shivang Agarwal, Jean Ogier Du Terrail, and Frédéric Jurie. (2018) Recent advances in object detection in the age of deep convolutional neural networks. CoRR, abs/1809.03193.

[57] Roger Kalliomäki. (2019) Real-time object detection for autonomous vehicles using deep learning. Master Thesis, Uppsala University.

[58] Olof Berg Marklund, Oskar Hulthén. (2019) 3D Object Detection for Autonomous Driving using Deep Learning. Master Thesis, Chalmers University of Technology.

[59] Konigshof, H., Salscheider, N.O., Stiller, C. (2019) Realtime 3d object detection for automated driving using stereo vision and semantic information. IEEE International Conference on Intelligent Transportation Systems.

[60] Jisoo Jeong, Seungeui Lee, Jeesoo Kim, and Nojun Kwak. (2019) Consistency-based semi-supervised learning for object detection. In Advances in neural information processing systems, pages 10758–10767.

[61] Alberto Garcia-Garcia, et al. (2017) A Review on Deep Learning Techniques Applied to Semantic Segmentation. arXiv:1704.06857.

[62] X. Feng, Y. Jiang, X. Yang, M. Du, X. Li. (2019) Computer vision algorithms and hardware implementations: A survey. Integration, vol. 69, pp. 309–320.

[63] Richard Szeliski. (2010) Computer Vision: Algorithms and Applications. Springer Science Business Media.

[64] Wilson Geisler. (1983) Vision: A Computational Investigation into the Human Representation and Processing of Visual Information. Psyccritiques, pp. 581–582.

[65] A. Krizhevsky, I. Sutskever, G.E. Hinton. (2012) Imagenet classification with deep convolutional neural networks. Adv. Neural Inf. Process. Syst. 1097–1105.

[66] Karen Simonyan, Andrew Zisserman. (2014) Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556.

[67] Christian Szegedy, et al. (2015) Going deeper with convolutions. Conference on Computer Vision and Pattern Recognition.

[68] Gao Huang, et al. (2017) Densely connected convolutional networks. Conference on Computer Vision and Pattern Recognition.

[69] Jie Hu, Li Shen, Gang Sun. (2017) Squeeze-and-excitation networks. Conference on Computer Vision and Pattern Recognition.

[70] Barret Zoph, et al. (2018) Learning transferable architectures for scalable image recognition. Conference on Computer Vision and Pattern Recognition.
[71] B. Zoph, Q.V. Le. (2017) Neural architecture search with reinforcement learning. International Conference on Learning Representations.
[72] Jonathan Long, Evan Shelhamer, Trevor Darrell. (2015) Fully convolutional networks for semantic segmentation. Conference on Computer Vision and Pattern Recognition.
[73] Liang Chieh Chen, et al. (2014) Semantic image segmentation with deep convolutional nets and fully connected CRFs. Comput. Sci. 357–361.
[74] Guosheng Lin, et al. (2017) RefineNet: multi-path refinement networks for high-resolution semantic segmentation. Conference on Computer Vision and Pattern Recognition.
[75] Hengshuang Zhao, et al. (2017) Pyramid scene parsing network. Conference on Computer Vision and Pattern Recognition.
[76] Navneet Dalal, Bill Triggs. (2005) Histograms of oriented gradients for human detection. Conference on Computer Vision and Pattern Recognition, vol. 1.
[77] Pedro F. Felzenszwalb, et al. (2010) Object detection with discriminatively trained partbased models. Trans. Pattern Anal. Mach. Intell. 1627–1645.
[78] Viola Paul, Michael J. Jones. (2004) Robust real-time face detection. Int. J. Comput. Vis. 137–154.
[79] Fei-Fei LI, Justin Johnson, Serena Yeung. (2018) Stanford online course: Convolutional Neural Networks for Visual Recognition. Lecture 5 - Convolutional Neural Networks. Stanford vision and learning lab. URL: http://cs231n.stanford.edu/syllabus.html.
[80] Schonberger, J. L., Frahm, J. M. (2016). Structure-from-motion revisited. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 4104-4113).
[81] Fuentes-Pacheco, J., Ruiz-Ascencio, J., Rendón-Mancha, J. M. (2015). Visual simultaneous localization and mapping: a survey. Artificial intelligence review, 43(1), 55-81.