TÉCNICO LISBOA

Vehicle Counting with Object Detection on Traffic Webcams

Ana Madalena Schclar Leitão

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. João Paulo Salgado Arriscado Costeira

Examination Committee

Chairperson: Prof. Alberto Manuel Rodrigues da Silva
Supervisor: Prof. João Paulo Salgado Arriscado Costeira
Member of the Committee: Prof. Maria Margarida Campos da Silveira

October 2020

Acknowledgments

I would like to thank my family, especially my parents, for their continuous support and encouragement over all these years. I would also like to thank my dissertation supervisor, Prof. João Paulo Costeira, for his guidance and insight throughout this project. Last but not least, to my friends, the ones I have known for years and years and the ones I met along this journey. Thank you for all the friendship and support.

Abstract

As the population residing in urban areas continues to grow, the need for understanding the traffic flow of a city is becoming increasingly more essential. Several cities have invested in the installation and deployment of Intelligent Transportation Systems (ITS). Some of the most commonly used sensors in ITS are traffic cameras used for analysis and monitoring purposes. Among the existing methods that can perform this analysis based on low-resolution camera footage, Object Detection is a compelling and yet to be specifically adapted approach.

In this work, we are using a state-of-the-art general object detector, Faster-RCNN [1], and adapting it to real-world low-resolution traffic footage. The traditional applications of this type of object detectors present considerable differences to our intended application on traffic data. These differences pose the main challenge of this approach and are mentioned throughout our work.

On the low-resolution traffic dataset we are using, our implementation of Faster-RCNN achieved a mean Average Precision (mAP) of 70%. Other state-of-the-art general object detectors achieved results 10% or 20% lower on the same testing conditions. In another traffic dataset, our approach achieved results comparable to those achieved by density estimation, a standard object and vehicle counting technique.

Keywords

Vehicle Counting, Object Detection, CNNs, Traffic Monitoring.

Resumo

À medida que a população residente em áreas urbanas vai aumentando, vai se tornando cada vez mais necessário o estudo e compreensão do trânsito das cidades. Muitas cidades têm investido na instalação de Sistemas de Transporte Inteligente (ITS) para este propósito. As câmaras de trânsito constituem um dos sensores mais comuns nas ITS e são utilizadas para fins de análise e monitorização. Dentro dos métodos capazes de analisar imagens de trânsito de baixa resolução, a Detecção de Objectos é uma abordagem de interesse e raramente adaptada a esta aplicação.

Neste trabalho vamos utilizar um detector de objectos de topo, o Faster-RCNN [1], e adaptá-lo a imagens de baixa resolução obtidas por câmaras de trânsito. As aplicações tradicionais deste tipo de detector de objectos apresentam fortes diferenças face à aplicação que aqui pretendemos efectuar. Estas diferenças causam a principal dificuldade desta abordagem e são mencionadas ao longo deste trabalho.

No dataset de trânsito de baixa resolução utilizado, a nossa implementação da Faster-RCNN obteve uma *mean Average Precision* (mAP) de 70%. Esta mAP é 10% ou 20% superior que a mAP obtida por outros detectores de objectos *state-of-the-art* nas mesmas condições de teste. Também realizámos testes noutro dataset de imagens de trânsito. Neste, a nossa implementação produziu resultados comparáveis aos obtidos através de calculo da densidade, uma técnica standard para a contagem de objectos, incluindo veículos.

Palavras Chave

Contagem de Veículos, Detecção de Objectos, CNNs, Monitorização de Trânsito.

Contents

List of Figures

List of Tables

Acronyms

ANPR	Automatic Number Plate Recognition
AP	Average Precision
CNN	Convolutional Neural Network
DPM	Deformable Part-based Model
FC	Fully Connected
FN	False Negatives
FP	False Positives
FPS	Frames per Second
GT	Ground Truth
HOG	Histogram of Oriented Gradients
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IoU	Intersection over Union
ITS	Intelligent Transportation Systems
mAP	mean Average Precision
RCNN	Region-based CNN
RoI	Regions of Interest
RPN	Region Proposal Network
SIFT	Scale-Invariant Feature Transform
SVM	Support Vector Machines
TP	True Positives

1

Introduction

Contents

With the ever-growing size of the world population, and particularly the increase of those living in urban areas, urban planning and management have become an essential part of any large metropolis. Streets, roads and traffic are one of the main components of a city, hence, its population can only achieve a satisfactory lifestyle if proper traffic flow conditions are maintained. For this reason, cities are placing a large interest in traffic monitoring and planning.

This interest has led to the development and evolution of Intelligent Transport Systems (ITS). The aim of ITS is to provide accurate information to road users, allowing them to make better use of the transportation networks.

Recently, research on ITS has been going in the direction of computer vision and machine learning. Both these fields have seen tremendous advances, reaching an unprecedented level of accuracy and speed in image analysis and object detection. However, the transition from general computer vision techniques to traffic monitoring applications is not trivial. These approaches require large amounts of training data to perform correctly and the traffic camera data is available is not extensive and varies considerably in terms of quality and resolution.

1.1 Goal

Our main goal with this work is to adapt a general object detection model to real-world traffic data. Through the application of object detection, our aim is to perform vehicle counting, one of the essential tasks of traffic monitoring.

The key challenge lies in the use of real-world traffic data. Most traffic cameras were not installed to be analysed through computer vision techniques, presenting serious problems to these approaches. The following are some of the major ones:

- low resolutions,
- cluttered and busy scenes,
- small vehicles,
- occluded vehicles.

Considering all the impediments above, we intend to evaluate if object detection is a feasible method of vehicle counting to be used with the existing traffic camera networks.

1.2 Contributions

Through the development of this work we have managed to:

- Adapt a general object detector to the task of vehicle counting through detection on low resolution images.
- Analyse CityCam, a traffic dataset with both urban and highway footage, and extracted the parameters relevant to the detection problem.
- Achieve results comparable to the more commonplace method of vehicle counting, density estimation.

1.3 Organization of the Document

This thesis is divided in six chapters. This first one briefly introduces the problem and our proposal. Chapter 2 provides background on the three main topics our work focuses on: traffic monitoring, object detection and vehicle counting. Here, we present techniques and state-of-the-art approaches used in each of these fields. In Chapter 3 we give a detailed description of the object detection architecture we used, Faster RCNN [1]. In this chapter we also present the two datasets used: CityCam [5] for training and testing and a dataset of Tallinn traffic footage for testing. Chapter 4 focuses on the adaptation of the object detection architecture to the traffic dataset we are using. Here we analyse the CityCam data and define our network parameters. In Chapter 5 we present all experimental results obtained in both the CityCam and the Tallinn data. Chapter 6 concludes this thesis with a discussion of the work developed and future work proposals.

2

Background

Contents

This chapter provides background on the three main topics this work focuses on: traffic monitoring, object detection and vehicle counting. On the subject of traffic monitoring we focus on traffic cameras and their applications. In the section of object detection we go over the evolution of detection architectures, from feature-based to deep learning. The last section, vehicle counting, presents the challenges of this task, counting systems based on object detection and one other technique, density estimation.

2.1 Traffic monitoring

The road transportation system is an essential component of modern society, and as such, its monitoring is a topic of great concern. This monitoring, and subsequent analysis, enables the improvement of road user experience, and most importantly, of user safety. This topic is a part of the Intelligent Transport Systems (ITS) field.

According to the European Commission "Intelligent Transport Systems and Services (ITS) refers to the integration of information and communication technologies with transport infrastructure to improve economic performance, safety, mobility and environmental sustainability" [6]. ITS aim to provide real-time information to road users allowing them to make better route decisions, reduce traffic congestion and improve the overall use of the transportation networks.

To perform the monitoring of the roads, ITS rely on several types of physical sensors, dispersed throughout the networks, that collect different kinds of data [7]. Early uses of traffic sensors depended on a human operator to observe the data output. However, in the last couple of decades, the tendency has been to process sensor output through automated computer systems, employing machine learning and computer vision techniques.

The following three sensors are the most common ones:

- Inductive Loop Detectors - they are the most commonly used sensor for vehicle detection. These consist of turns of wire embedded in the pavement and connected to an exterior control unit. They work by detecting changes in their magnetic field when a vehicle passes over them, allowing vehicle count, classification and, by using pairs of loops with small distances between them, it is possible to measure vehicle speed. Although these are the most prevalent sensors, they present serious shortcomings. The installation of these sensors requires modifications to the pavement and lane closure, also, they tend to fail and break down quite frequently, having high maintenance costs.

- Microwave radar detectors - these are used for vehicle counting and measuring speed. They are usually installed above or next to the road providing a view from above. Unlike inductive loops, these don't require pavement modifications, which makes them non-intrusive detectors. Although

the information they provide is not as complete as that obtained by using loop detectors, these sensors have the advantage of working under all weather conditions.

- Traffic cameras - These sensors offer the richest information on traffic. From camera footage, we can obtain vehicle counts, classifications, measure speed, detect incidents and traffic congestion. They are non-intrusive detectors, mounted in structures similar to those of microwave detectors, having much lower installation costs than inductive loops. Traffic cameras have the disadvantage of being heavily influenced by weather and lighting conditions. When the analysis of camera footage depended on human operators, these sensors were a very costly option, however, with the recent advances in computer vision and image processing, cameras have become one of the most researched sensors for traffic monitoring.

Besides the ones listed above, there are other less common sensors. Infrared, ultrasonic and acoustic detectors are non-intrusive and similar to microwave radar detectors. Magnetic and pneumatic tube sensors are intrusive detectors that work similarly to inductive loops.

Transportation systems simultaneously employ an assortment of these sensors, combining all the information collected. Sensors are spread out throughout the roads to ensure full coverage both in time and geographically. They are selected based on installation cost and the type of information required.

In some cases, road users and traffic police also provide information which can help augment the quality and scope of the data.

In Intelligent Transportation Systems, data obtain by traffic monitoring is used for three different purposes, real-time information, capacity information, and prediction. The first is the supply of real time traffic volume, vehicle speeds, incident occurrence and journey times to users. The second is the modelling of the road network in terms of capacity. The third is the prediction of future traffic conditions. To achieve this last one, analysts apply data mining techniques to high volumes of stored data, accurately time-stamped and sourced, obtaining relevant traffic patterns on the network.

With this work, our focus is on traffic footage collected by cameras. Therefore in the rest of the section we will further explore this sensor, introducing the different types of traffic cameras and their applications in ITS.

2.1.1 Traffic Cameras

For a long time, transport systems have used cameras for traffic monitoring purposes. They provide rich contextual information for human understanding and have relatively low installation and maintenance costs.

Early uses of cameras required a human operator to observe the traffic footage and manually extract data. As a result, cameras were an impractical source of data to monitor an entire road network, being

mainly used as a complementary sensor.

However, computer vision and video processing techniques have seen extreme advances in the past two decades, enabling the automatic extraction of information from traffic camera footage. For this reason, cameras are now one of the most researched data sources in ITS.

Camera types

Cameras serve a variety of purposes in ITS, having distinct requirements. In [8], the authors divide traffic cameras based on footage resolution, presenting the following three categories:

- Traffic webcams - low resolution and low frame-rate cameras. Their primary purpose is to inform road users on the current traffic state. They are the least costly option and cover both urban and highway scenes. The footage displays an extensive stretch of road and a large number of vehicles per frame. The resolution of these cameras is not sufficiently high as to allow the recognition of license plates or other vehicle details.

- Traffic surveillance cameras - these cameras have a higher resolution and frame-rate than the ones before, however, they are still unable to capture exact details like license plates. They can accurately display a section of road and are mainly used for incident detection, being prevalent in areas such as tunnels and bridges.

- High-detail cameras - high-resolution cameras that precisely capture a vehicle's license plate and driver. These tend to have a very restricted field of view, observing only one vehicle per instant. The most common application for this type of camera is ANPR. ANPR is mainly applied in toll stations on highways for fee-charging purposes.

While high detail cameras and their applications are a quite relevant area of ITS, they are not in the scope of this thesis. Hence, the rest of the section is only concerned with traffic webcams and surveillance cameras.

(a) webcam **(b)** surveillance **(c)** high-detail

Figure 2.1: Examples of images from the three traffic camera types.

Both these cameras are usually mounted on high structures, such as traffic lights and poles over the

road. In highways and freeways, they are installed in a way that permits the observation of a long stretch of road in focal parts of the network. In urban areas, they are mainly located at intersections.

There are a few different types of camera installation layouts. Monocular cameras, i.e. single-lens cameras, are the most common, these provide a 2D image of the scene. In urban areas, there is usually no more than one camera per intersection. However, there are some locations with multiple cameras pointing at the same scene. This layout provides 2D footage from different viewpoints making it possible to recreate a 3D scene. Another type of camera is stereo cameras, i.e. double-lens cameras. They provide depth information which also allows the construction of a 3D scene.

Applications

In ITS, surveillance type cameras and webcams are mainly used for analysis purposes. These include the following:

- Vehicle counting - To understand a region's traffic, it is essential to know how many vehicles occupy the roads and at what times. Vehicle counting provides the information needed to understand the use and capacity of the current road infrastructure and also to plan new roads. Large quantities of data are required for this purpose and they tend to be analysed offline. Counting can be performed by using object detection techniques or more commonly by density estimation.

- Traffic incident detection - One of the main concerns of ITS is the safety of road users. The goal of this application is to reduce accidents by detecting dangerous situations, such as stopped vehicles, vehicles going in the opposite direction, obstacles on the roads, etc. For incident detection, the analysis has to be performed quickly so that authorities and road users are promptly informed.

- State recognition - This application informs users on the current traffic situation, such as congestion and dense traffic on certain parts of the network. This information can be obtained by vehicle counting, however, for state recognition, counts are not required to be as exact. Low-resolution video is usually enough to provide the necessary information.

2.2 Object Detection

Object detection is one of the fundamental tasks of computer vision. It is the task of determining whether an object of a particular class is present in a given image and returning its spatial location. This task can be seen as a step further from image classification. Classification is the task of specifying what object or objects are represented in an image, without the need to offer their positions, size or number of occurrences.

Research in this area aims to develop computational models that identify and locate all objects from multiple classes in images. Some works focus on specific problems, such as pedestrian detection, face

recognition or vehicle detection, etc. Other works attempt to solve generic object detection, that is, detecting objects from diverse classes trying to simulate human vision.

To represent the spatial location of an object it is common to use either a bounding box (represented by a rectangle's vertices) or a segmentation mask (pixel-wise partitions). Figure 2.2 shows an example of generic object detection with bounding box predictions. When using bounding boxes, the correctness of a detection is given by the Intersection over Union (IoU) score. This is a measure of the overlap between two bounding boxes: the ground-truth (GT) annotation and the prediction.

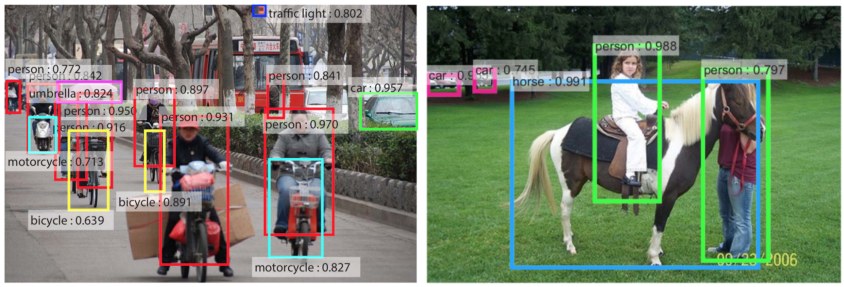


Figure 2.2: Example of object detection results from [1], obtained with Faster RCNN detector. On both images, bounding box predictions represent the location of the objects. The class prediction and the associated confidence score are placed on top of the box frame.

Challenges

The task of generic object detection faces several challenges both in terms of accuracy and efficiency. We can divide accuracy issues into two categories: intra-class variations and inter-class distinctions.

- Intra-class Variations - detectors have to be robust to the many transformations that can occur between different instances of the same class. Figure 2.3 shows an example of some of these variations in the *car* category. These transformations can be further divided into two:
  - Intrinsic Factors - differences in the characteristics inherent to the objects. For example variations in sizes, colour, texture, material, etc.
  - Imaging Conditions - characteristics that depend on environmental and surrounding conditions. These problems arise from the characteristics of real-world images, due to images being digital 2D projections of 3D world scenes. The factors that cause them can be different cameras, illuminations, weather, viewpoints and backgrounds. All of these will affect an object's pose, scale, illumination, sharpness, partial occlusion, and more.
- Inter-class Distinctions - detectors are required to solve ambiguities between classes and differentiate the large number of object classes present in the real-world.

Besides these, cameras also produce image noise and distortions which affect the detection task.

Figure 2.3: Example of four different car images in the Imagenet dataset [9]. These examples are taken from different angles and have different backgrounds (imaging conditions) and the *car* objects have different shapes and colors (intrinsic factors).

In terms of efficiency, the demands arise from the always increasing number of images and data. On the application of a detection algorithm, there will be a need to find the location of many objects, belonging to several classes, across several images. All this, while having constraints in terms of time, memory and processor capacity.

There has been an interest in the task of object detection and more generally, computer vision, for many decades. However, earlier on, computers were not equipped to efficiently solve such tasks. Due to the incredible improvements in computer technology, the most notable advances in object detection research have occurred in the last two decades. Many comprehensive reviews have been published on this topic [10, 11]. Generally, the advances are divided into two eras: traditional feature based detectors and deep learning based detectors. In the following sections we explore the historic progress of these two techniques.

2.2.1 Traditional Detectors

During this era, object detectors were built based on handcrafted features. A feature, also referred to as keypoint, is a part of an image of particular interest to a recognition task, usually a point with special visual characteristics. In other terms, a feature is an image region with maximum intensity variation when moved by a small amount. A set of these simple local features is an image representation that allows the recognition of a certain object.

The concept of extracting local features for object detection was inspired by the way animals process visual stimuli [12]. They begin by responding to basic features, such as oriented edges, and from those build up more complex structures until the brain can recognise real-world objects. Detection based on feature extraction follows a similar logic. First, an algorithm discovers features and computes descriptors, then, it produces a representation for the image and finally, it employs machine learning to classify the image representation. Figure 2.4 shows a basic representation of a traditional feature based detector.

Several feature representations and detectors have been proposed throughout the years, below we focus on some of the main ones:

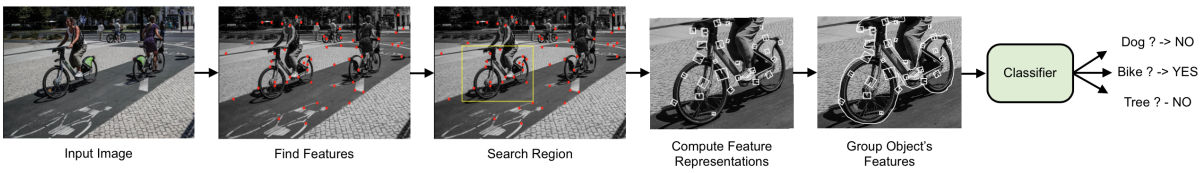


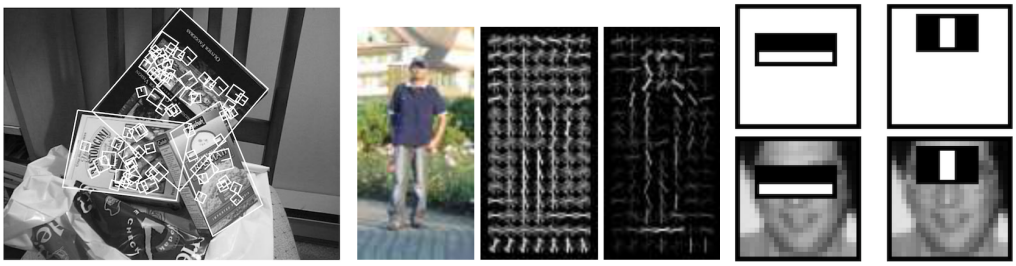
Figure 2.4: Representation of a basic feature detection algorithm.

In 1991, Lowe presented Scale-Invariant Feature Transform (SIFT) [13]. This feature representation was one of the most influential publications in feature-based detection. This method begins by building an image pyramid, ensuring scale invariance. At each scale, or pyramid level, the original image is smoothed with a Gaussian function to create increasingly more blurry images which are then subtracted from the sharper ones. This subtraction discards high-frequency components, that represent noise, and low-frequency ones, that represent homogeneous regions, locating the edges and corners. A nearest neighbour approach is used to find local extrema on these subtracted images, across different scales. These local extrema are potential keypoints. Low-contrast and edge keypoints are eliminated as potential keypoints, leaving only high interest points. For each keypoint, an orientation is assigned, this corresponds to the main gradient orientation present in the pixels surrounding the keypoint. Next, having already the keypoint's scale, orientation and location, a local image descriptor is created. This is a fixed sized vector representing a 16x16 pixel patch centred on the keypoint. The values forming the vector correspond to the bins in the histogram of the gradient orientations in that image patch. Figure 2.5 shows an example of SIFT keypoints.

In 2001, Viola and Jones [14] published a major work in object detection. They developed a face detector that uses Haar-like features. These are essentially groups of 2, 3 or 4 rectangles in particular arrangements, shown in Figure 2.5. The pixel intensities of each rectangle are summed and, depending on the group layout, the sums of some rectangles are subtracted from the other sums. To detect faces they use the sliding window technique to extract features from all locations. By using AdaBoost, they found that a small set of Haar-like features can be used to accurately classify a face (Figure 2.5). To improve efficiency they use cascade classifiers, first most windows are rejected by simple classifiers, the rest are then further processed and rejected by increasingly more complex classifiers.

In 2005, Dalal and Triggs presented Histogram of Oriented Gradients (HOG) features [15]. These can be seen as an improvement on SIFT. Orientations are computed on a dense grid of uniformly spaced cells instead of being computed in keypoint patches. To detect objects they use a sliding window technique and apply a filter at all positions and scales of an image, the patches of HOG features are classified with linear SVMs. The HOG detector was developed primarily for pedestrian detection but has since been used in numerous applications. Figure 2.5 shows an example of HOG descriptors.

One of the last main contributions before the deep learning era was the Deformable Part-based



(a) SIFT **(b)** HOG **(c)** Viola and Jones

Figure 2.5: Example of three image feature representations, on the left, SIFT [13] keypoint features, on the middle, HOG [15] oriented gradients (the rightmost image shows the HOG descriptors after the application of SVM) and on the right, Haar-like features [14]. All figures were obtained from the respective papers.

Model (DPM) [16, 17], proposed in 2008 as an extension to the HOG detector. An object's model is made up of one root filter, several part filters and the deformation models. On a feature pyramid (feature maps at different scales), the root filter is applied as in the HOG detector. The object's model is scored at the root filter's location and scale as the sum of the root's score and the part filters' scores. A part's score is the maximum score of a part filter at different locations minus the deformation cost. This cost represents the difference between the part's location and it's ideal location relative to the root. The model's parameters are learned with latent SVMs.

2.2.2 Deep Learning Based Detectors

The second era of object detection begins in 2012 with AlexNet, a deep convolutional neural network from Krizhevsky et al. [18]. This network achieved incredible results on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [9] classification task. Due to those results, the use of deep CNNs went from image classification to object detection with Girshick et al. and their RCNN (Region-based CNN) in 2014 [19]. It was found that deep neural networks can reach astounding accuracy results in detection tasks. This influenced the development of several new detectors based on deep learning.

This era of object detection is generally grouped into two categories: two-stage detectors and one-stage detectors. The former begins by dividing an input into different regions before classification while the later perceives detection as a single step, that is, localisation and classification performed simultaneously. It is to note that this field evolves at an incredible speed and the state-of-the-art is constantly changing.

Backbone Networks

Deep learning based detectors depend on feature extraction networks, also referred to as backbone network. At first, these networks were CNNs design to perform image classification. Now, some are specifically devised for detection. The idea behind them is similar to the feature extraction method

of traditional detectors. First they obtain features, and then apply a classifier. Figure 2.6 shows a representation of VGG16 [2], a backbone CNN. The convolutional layers extract the features, the fully connected layers compute a representation vector and the last layer applies a softmax function to the vector, resulting in class probability scores.

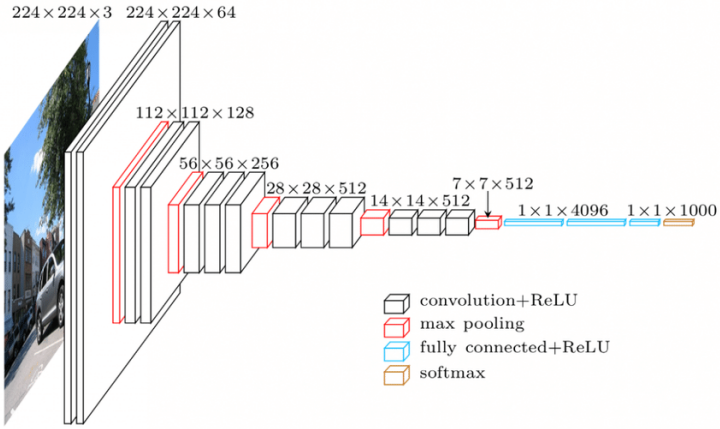


Figure 2.6: Representation of a backbone CNN: VGG16 [2]. Diagram from [3].

The introduction of AlexNet [18] in 2012 marks the beginning of backbone CNNs. This is an eight-layer network, with five convolutional layers and three fully connected (FC) ones. It achieved record-breaking results in the image classification task of the ImageNet LSVRC2012 [9] (15,3% error rate vs 26,6% for the second place). After, the tendency has been to increase the depth, i.e the number of layers, with each new network.

The evolution of CNNs and their performance can be shown through the results of the ILSVRC classification task over the years. In 2013, ZFNet [20] won with 14.8% top-5 error, this network is quite similar to AlexNet, maintaining most of the architectural aspects, including the number of layers.

GoogLeNet (or Inception net) [21] won in 2014 with 6.67% top-5 error, it has a total of 22 layers and no FC ones. While it goes deeper than the previous networks, it has incredibly fewer parameters due to the small size of the convolution filters used.

VGG net [2] appeared slightly before GoogLeNet and placed second in 2014 with 7.4% error, however, it has been one of the most widely used CNN for feature extraction, used in [1, 22]. It is made up of 16 convolutional layers and has a very uniform and simplistic architecture, [2] is a very influential work on reinforcing the idea that network depth is beneficial to the feature representation of an image and its consequent classification.

In 2015, ResNet [23], or Residual Neural Network, won, achieving 4.5% error rate, which is better than human-level error rate. This network builds up on the idea that the deeper the better, having up to 152 layers. This level of depth is made possible by the introduction of Residual Blocks, these can be imagined as connections that allow skipping layers or shortcuts between layers.

In 2017, DenseNet [24], with up to 264 layers, introduced a block that connects each layer to every other in a feed-forward fashion. In the same year, SENet [25], won the challenge with a 3.79% error rate by integrating a block that learns channel-wise feature responses.

Most modern detectors employ these CNNs, or their extensions. Furthermore, some detectors can interchangeably apply more than one backbone network.

Two-stage Detectors

Also known as region-based approaches, these detectors begin by generating class-independent region proposals from an input image. Next, a backbone CNN extracts features from the proposed regions and a classifier determines the category of each proposal. The output is a set of labeled regions, i.e, detected objects. Figure 2.7 contains representations of two famous two-stage detectors, RCNN [19] and Faster RCNN [1].

In 2014, Girshick et al. published a breakthrough work in deep learning and object detection, RCNN (Region-based CNN) [19]. This architecture is made up of three modules, represented in Figure 2.7. The first uses selective search [26] to obtain around 2000 region proposals. Each proposal is warped to a fixed dimension and fed through a backbone CNN (AlexNet [18] in the original implementation), resulting in a feature vector. The third module consists of class-specific SVMs that score each feature vector, predicting the presence of objects and classifying them. The final phase of this architecture is bounding box regression to adjust the predicted boxes and non-maximum suppression to reject duplicate proposals. RCNN obtained an unprecedented 58.5% mean Average Precision (mAP) on the Pascal VOC07 challenge [27]. Nonetheless, it has a major drawback, there is a considerable amount of overlap between the region proposals, leading to redundant computation of features. This causes slow detection speed (around 14s per image).

In the same year, He et al. proposed SPPNet (Spatial Pyramid Pooling Network) [28]. This architecture includes a Spatial Pyramid Pooling layer. This layer is able to generate a fixed-length feature representation independently of the input image size. With this, the redundant feature computations of RCNN are eliminated and a feature map of the entire image is generated only once. From this feature map, the SPP layer computes fixed-length representations of different-sized arbitrary regions. SPPNet introduces a considerable speed-up from RCNN, being over 20 times faster while obtaining 59.2% mAP on VOC07.

In 2015, Girshick proposes Fast RCNN [22], an improvement on his previous work [19]. This method simultaneously learns to classify object proposals and refine their spatial locations. It was the first method to enable fine-tuning of all its layers. As feature extraction CNN it uses VGG16 [2]. Fast RCNN obtained 70% mAP VOC07 and processes an image in 0.3 seconds, excluding object proposal time.

After Fast RCNN, the next step was to perform region proposals solely with CNNs. This was accomplished in the same year by Ren et al. with Faster RCNN [1]. This method introduced the Region

Proposal Network (RPN), a module that generates nearly cost-free region proposals. Figure 2.7 shows a representation of this architecture. It uses the same feature map computed for classification which allows a much faster processing speed, around 5 images per second with VGG16. Faster RCNN obtained 73.2% mAP on VOC07 and was the first detector to enable complete end-to-end training. Further improvements to Faster RCNN were proposed, such as RFCN [29] and Mask RCNN [30]. Both use the much deeper ResNet [23] and achieved results of around 80% mAP on VOC07.

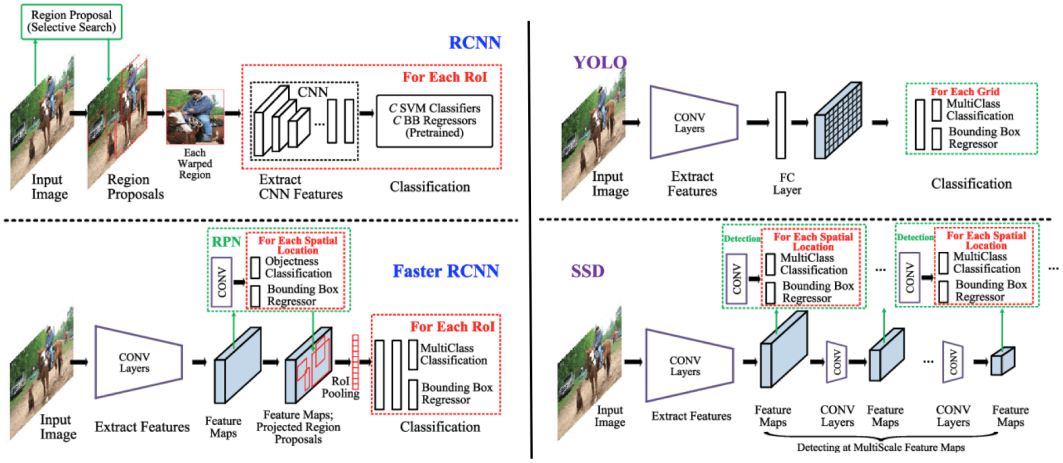


Figure 2.7: Schematic representations of four object detectors, from [11]. On the left are the two-stage detectors RCNN [19] and Faster RCNN [1]. On the right are the one-stage detectors YOLO [31] and SSD [32]. These representations highlight the difference between both types of architectures. Two-stage detectors are evidently modular, while one-stage detectors show a much more streamlined process.

One-stage Detectors

One-stage detectors arose from the desire to create faster architectures that were not hindered by a region proposal module. Thus creating systems with a unified framework. In this type of architecture, class probabilities and bounding boxes are predicted simultaneously, directly from the full image's feature map. Figure 2.7 shows representations of two one-stage architectures, YOLO [31] and SSD [32]. Detectors with this one-stage approach, also referred to as single shot detectors, obtained incredible speed-up results when compared to the state-of-the-art Faster RCNN. However, they display some difficulty in detecting small objects.

One of the first one-stage detectors was proposed in 2015 by Redmon et al., YOLO (You Only Look Once) [31]. It divides an input image into regions and a single CNN predicts bounding boxes and class probabilities for each region, having to only look once at an image to predict what and where objects are. This makes it much faster than previous detectors, being capable of processing images in real-time at 45 frames per second. However, this speed-up came at the compromise of accuracy. YOLO achieved 66.4% mAP on VOC07 (vs Faster RCNNs [1] 73.2%). This drop in detection accuracy is especially evident for small objects.

Later in the same year, Liu et al. proposed Single Shot Multibox Detector (SDD) [32]. SSD improves detector accuracy by obtaining predictions at different scales on different layers of the network. Until this point, detectors only applied classifiers to the final feature map. This multi-scale technique allows for a better detection of small objects. SSD achieved 76.8% mAP on VOC07 while running at 59 frames per second.

After SSD, two improved versions of YOLO [31] were proposed, YOLOv2 [33] and YOLO9000 [34]. The first increased the results on VOC07 to 78.6% while the second introduced detection at different scales to improve detection of small objects.

Datasets and Benchmarks

An essential part of developing a successful deep learning detectors is the quantity and quality of the data used for training. As such, the performance achieved by detectors is made possible by the existence of large and extensive labeled datasets.

The following are the most widely used generic image datasets for training detectors and classifiers:

- PASCAL VOC - this dataset is associated with the PASCAL VOC challenge [27], a visual recognition challenge that was held from 2005 to 2012. Most of the results presented by researchers upon publishing papers are obtained according to this challenge's guidelines. The train/validation dataset used in 2012 contains 11530 labeled images and objects belonging to 20 different classes.
- ImageNet - this is one of the largest visual databases. It uses crowdsourcing to obtain bounding box and class annotations on images. The ILSVRC (ImageNet Large Scale Visual Recognition Challenge) [9] uses a small part of this database as a dataset. The competition started in 2010 and the latest edition uses 150000 labeled images with 1000 classes as validation/test data and a random 50000 images sample as training data.
- Microsoft COCO - MS COCO [35] is one of the latest benchmark datasets. It is composed of more than 200000 images with labeled objects belonging to 80 classes. COCO stands for Common Objects in Context, the goal with this dataset is to have detectors use context as a part of their prediction. A competition is also held with this dataset since 2015 and it is the most recent standard for detector evaluation.

2.3 Vehicle Counting

With the growing interest in ITS and road monitoring, vehicle counting and its methods have also been attracting more attention. There are two main methods to count vehicles in camera footage: detection and density estimation. In this section, our primary concern is vehicle detection, its characteristics and methodologies. In the final part of the section we briefly discuss vehicle density estimation.

Vehicle detection is a specific application of object detection. Essentially, it is a system that analyses input traffic photos and provides as output the locations and classes of vehicles in the image.

Challenges

As a task of object detection, vehicle detection faces most of the same general challenges, mentioned in section 2.2, and several more in addition. Below, we begin by comparing this type of detection and the general one. After, we present the main issues that arise from environmental conditions.

The most significant difference between general object detection problems and traffic monitoring problems is the type and quality of the data. They are the following:

- Image Resolution - In object detection, images tend to be of high-resolution. In vehicle detection, some applications also use high-resolution cameras, however, the footage is more often obtained by surveillance type cameras or traffic webcams. These last ones especially produce footage of low-resolution and low frame-rate, making features harder to determine.

- Number and Dimension of Objects - In general detection problems, images commonly feature only a couple of foreground objects with large dimensions. In contrast, traffic images present very dense scenes. Traffic cameras are intended to capture a long section of the road and as many vehicles as possible. Consequently, vehicles appear at very different scales. The biggest challenge is that a substantial number of vehicles is captured far away from the camera, having extremely small dimensions in the images.

- Level of Occlusion - Due to the layout of traffic scenes and the number of vehicles featured, it is quite common for vehicles to appear occluded by others. Some of these occlusions are quite severe, especially in photos of heavy traffic where the vehicles closest to the camera almost completely cover the rest. In general detection, datasets tend to present low levels of occlusion in their objects.

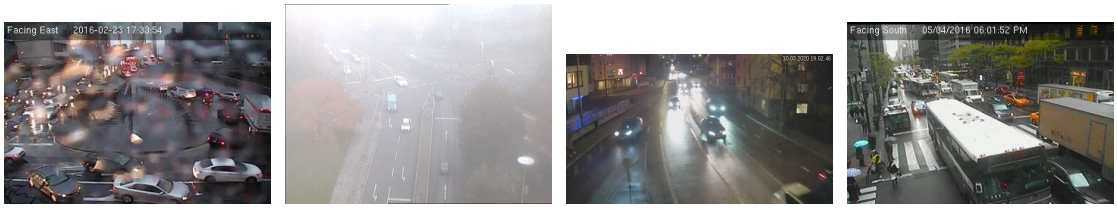
- Class Distinction - The type of class distinction desired for vehicle detection presents another crucial difference. Unlike generic object detection, vehicle detection is concerned with only two main classes, vehicles and, less often, pedestrians. However, the small distinctions between vehicle subclasses are relevant in traffic monitoring. This level of discrimination is not necessary in most general detection systems.

In terms of environmental conditions, in [36] the authors present a review of video-based ITS systems with a focus on the environmental factors that affect vehicle detection. They specify two main factors that influence vehicle recognition (Figure 2.8 illustrates some examples) :

- Illumination - Daytime and nighttime traffic images are quite different. During the daytime, cameras record vehicle shape, texture and colour, all three characteristics are used as features in vehicle

detectors. At night, the most distinctive details are the headlights and taillights, making these the features preferred for detection. It is quite challenging to design a vehicle detector that is accurate in both daytime and night illuminations. The typical approach to this issue is to focus on only one of these two illumination conditions. Another hindering illumination factor is shadows, these can also substantially alter vehicle appearance.

- Weather - While illumination changes throughout the day are predictable, weather conditions are fluctuating. Harsh weather conditions, such as heavy rain, snow and fog, are detrimental to the quality of the video captured by cameras, making detection more challenging. Some studies conduct tests under different weather scenarios. However, the norm is to create systems for clear or mild weather.



(a) rain **(b)** fog **(c)** nighttime **(d)** occlusion and clutter

Figure 2.8: Example images of factors that affect traffic footage.

As mentioned in section 2.2, object detection is divided into traditional feature-based detectors and deep learning based detectors. The same distinction is applied to vehicle detection, as such, we will now go over both of these types of detectors.

2.3.1 Traditional Vehicle Detectors

As in general object detection, most of these methods divide into two steps: feature extraction and classification. In reviews of vehicle detection systems [36, 37], the authors tend to refer to these steps as Hypothesis Generation (HG) and Hypothesis Verification (HV). The HG step generates candidate vehicle locations. The HV step tests the locations to verify the presence of a vehicle. This two-step approach avoids the necessity of extensively searching an entire image for vehicles. Only the locations one is more likely to occupy are examined.

For the most part, we can categorise HG methods into two, appearance-based and motion-based, in [38] these are referred to as bottom-up and top-down, respectively.

Appearance-based

Also called knowledge-based, these methods make use of known features and information about vehicles to recognise them. In other words, they employ *a priori* knowledge to detect vehicle locations.

Studies use several distinct features to create efficient vehicle descriptors. Some approaches, especially the earlier ones, make use of simple and easy to extract low-level features. Examples of these are symmetry, colour, shadow and edge information. Each has a different application: colour is favourable to the detection of headlights; symmetry is implemented when the vehicles are in frontal or rear-view; shadows are detected and eliminated to avoid miscalculations of the shape and location of vehicles; edge extraction provides contour information.

Although useful, these individual features are usually not sufficient to represent all the necessary information. To address this, some studies combine multiple features or, as of more recently, use local feature descriptors. Local feature descriptors are representations designed to identify objects in an image and to handle object and scene variations. Some of the more common ones applied to vehicle detection are Histogram of Oriented Gradients (HOG) and Haar-like features, as in [39] and [40], respectively.

Motion-based

Motion-based methods take advantage from the fact that vehicles on the road are usually moving. These are methods that extract moving objects from a static background. With no prior knowledge they can distinguish a vehicle, based on motion. Unlike the previous methods, these usually require a sequence of images.

The most basic approach is based on frame differencing: a difference map is obtained by subtracting previous frames from the current one and thresholding the resulting intensity values; The highest intensity pixels are then considered as vehicle proposals. This technique alone tends to be lacking and is more usually employed as a first-step in a detection algorithm [41].

A more complex version of the previous approach is background modelling where a model is built for an entire video stream and not for each sequence of frames. This method is usually performed in three steps: background initialization, foreground detection and background up-date. The Gaussian Mixture Model (GMM) [42] is one of the most popular techniques of background modelling for vehicle detection [43].

The choice between these two hypothesis generation methods depends on the application. Appearance-based methods tend to require a fixed view of the vehicle (usually rear or front view) and it is difficult to design appropriate features and descriptors. However, these methods are fast and easy to implement (having the features). Motion-based methods are more time consuming and require a video stream, not just frames. Yet, these are less affected by illumination and background changes.

As for the HV, or classification, step. It is like that of any other traditional object detector. Vehicle detection systems tend to use SVMs and AdaBoost classifiers, in [44] the authors compare the use of both. Some use simple Neural Networks as classifiers [45].

2.3.2 Deep Learning Vehicle Detectors

In vehicle detection, researchers have mainly focused on applying deep learning detectors to autonomous driving problems. In general, a detector, such as Faster RCNN [1] and YOLO [31], is trained and tested on a high-resolution dataset, captured with on-vehicle cameras. The main benchmark for these systems is the KITTI dataset [46] and very successful results have been achieved. In [47] Faster RCNN is trained and reported on KITTI and in [48] it is applied to Stanford Cars Dataset [49].

Regarding traffic camera footage, in [50] the authors propose a three module system to track and efficiently query the obtained data, using YOLOv2 [33] as the detector. They use their own traffic camera data (720p resolution) to train and report results on vehicle counts. In [51], the authors present a new highway traffic dataset with vehicles at very distinct scales. The system they propose begins by dividing the road area in the video into two, the proximal area and the remote area. Both areas are separately submitted through YOLOv3 [34] detector. The obtained vehicles are merged and the ORB algorithm [52] is used to obtain features from each object. These features are used to track vehicles and compute their trajectories in the video stream.

In general object detection we can see that the tendency for the last six years has been to use deep learning and construct better and faster detectors using deep CNNs. However, for vehicle detection, there are still several studies based on traditional detection techniques being proposed. This is especially evident for works focusing on low-resolution traffic camera data, and less so for autonomous driving purposes.

2.3.3 Counting with Density Estimation

Another popular technique to count vehicles in images is through density estimation. With density estimation, it is unnecessary to detect individual objects. The idea is to find a mapping between local image features and the corresponding density map. On said map, the count of vehicles is obtained by summing the pixel values.

In [53], a pioneering work in density-based counting, the density function is modeled as a linear transformation of a pixel's feature vector. As explained in [53] "Given the estimate F of the density function and the query about the number of objects in the entire image I, the number of objects in the image is estimated by integrating F over the entire I." Since this first work in 2010, density estimation

methods have evolved to CNNs and fully convolutional networks (FCN). Figure 2.9 shows an example of an image an the corresponding density map.

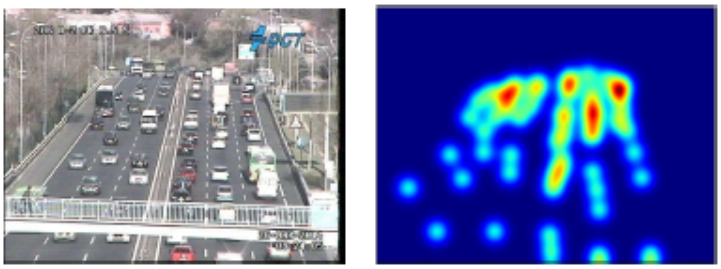


Figure 2.9: Example of a traffic image and the corresponding density map, from [4].

In the domain of vehicle counting, Guerrero-Gomez-Olmedo et al. [4] analysed two counting by density models [53, 54] on their dataset, TRANCOS, a dataset especially designed for counting severely overlapped vehicles.

In 2017, Zhang et al. [5] published a work focusing on low-resolution traffic webcam data. They propose two solutions to the problem. The first solution is an improvement based on [53], where they introduce geometry information by dividing a target region into blocks and learning different weights per block. With this block system they can sidestep the issue of vehicles at severely different scales in traffic footage. Their second solution is a FCN model that simultaneously learns vehicle density and vehicle count. First they extract the features through a typical convolutional network, after, to produce a density map with the same dimensions as the input, they use deconvolutional layers. The joint density estimation and count cuts down the error associated to counting as a sum of the density. This error is particularly large in the presence of extreme occlusion or oversized vehicles. Both these solutions achieved positive results and adapt well to different vehicle scales.

Table 2.1 contains a summary of the vehicle counting/detection systems mentioned in this section.

Table 2.1: Overview of studies on vehicle detection/counting.

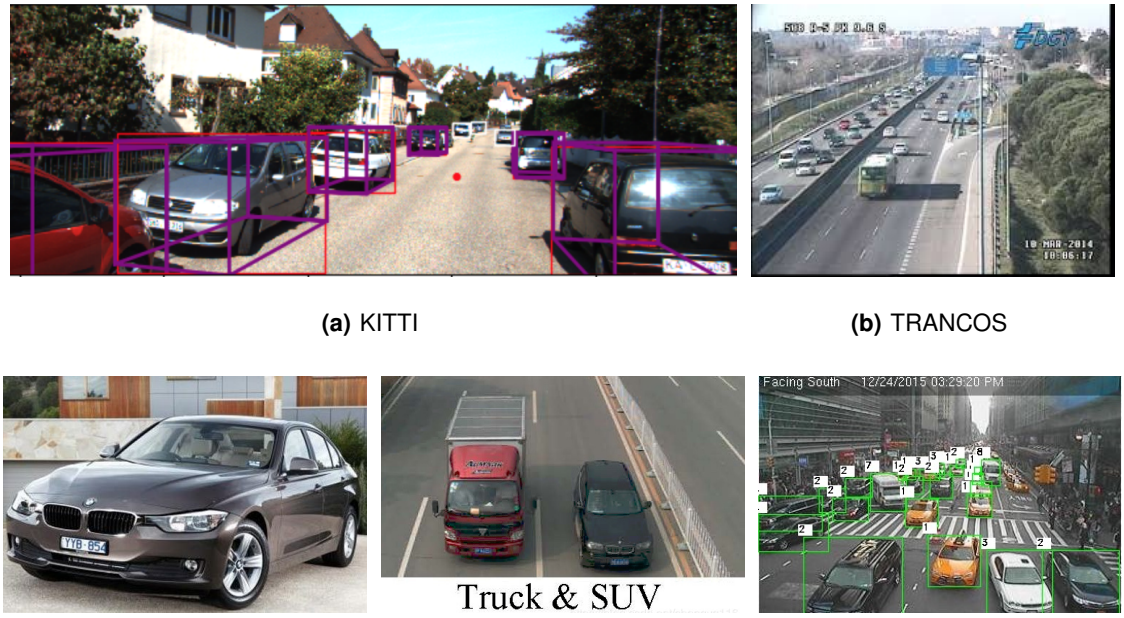
paper	method	type	model	data type	camera type	year
[39]	detection	feature-based	HoG+AdaBoost	med-res	on-vehicle	2016
[40]			Haar+SVM	med-res	on-vehicle	2014
[41]			HoG+SVM	low-res	traffic cam	2016
[43]			HoG+SVM	low-res	traffic cam	2012
[45]			Haar+NN	low-res	-	2015
[47]		deep-learning	Faster RCNN	high-res	on-vehicle	2016
[48]			Faster RCNN	high-res	on-vehicle	2019
[50]			YOLOv2	med-res	traffic cam	2017
[51]			YOLOv3	high-res	traffic cam	2019
[4]	density	feature-based	HoG	low-res	traffic cam	2015
[5]		deep-learning	-	low-res	traffic cam	2017

Datasets

As an application of object detection, datasets play an extremely important role in the development of efficient vehicle detection systems. However, there isn't an abundance of traffic data available as there is for generic object detection.

Although traffic surveillance cameras are used worldwide, the entities that control them rarely release the data publicly. This is mainly due to privacy and security reasons. The most widely available type of data comes from traffic webcams, whose quality is not high enough to recognise citizens identifying elements, and thus don't pose a privacy issue. Besides this, the data that exists is not extensively annotated.

The following are some of the largest vehicle detection datasets, it is to note that some do not have data from traffic cameras, but from on-vehicle cameras or other non-monitoring cameras.



(a) KITTI **(b)** TRANCOS

(c) Stanford Cars **(d)** BIT-vehicle **(e)** CityCam

Figure 2.10: Example images from the five vehicle datasets mentioned.

- KITTI [46] - This is one of the largest vehicle datasets for computer vision applications. It was created for the development of autonomous driving systems and thus contains data from on-vehicle cameras. The footage is of high-quality and represents real-world traffic scenes. The detection data consists of 15000 images with around 80000 labels (cars, vans, trucks, trams, pedestrians and cyclists). For their evaluation benchmark they define three levels of detection difficulty. These depend on bounding box height, level of occlusion by other objects and truncation my image margins.

- TRANCOS [4] - It is a dataset created for vehicle counting in traffic congestion situations. It consists of 1244 highway images with almost 47000 annotated vehicles. The annotations are for counting and not detection. They are not bounding boxes but rather dots localising the vehicles.
- Stanford Cars [49] - This dataset contains over 16000 high-resolution photographs of cars. It was created for the purpose of classification, categorising cars by make and model, although it does contain bounding boxes. Each image has only one vehicle that occupies most of the area.
- BIT-vehicle [55] - This dataset contains almost 10000 high-resolution vehicle images. Each image contains only one or two vehicles from the following categories: bus, microbus, minivan, sedan, suv and truck.
- CityCam [5] - It is a traffic webcam dataset composed of around 60000 images with almost 900000 annotations. It depicts real-world traffic scenes, both urban and highway, with vehicles categorised into 10 different classes.

Although there are a few datasets, there isn't a widely used benchmark. Vehicle detection systems are very dependent on the application and a lot of publications are done using specific and private datasets.

3

Vehicle Counting by Detection

Contents

This thesis focuses on the traffic monitoring task of vehicle counting through detection. The goal is to apply a well-known general object detection architecture to webcam traffic data.

In this chapter we will present the detection architecture and the datasets used. as well as the metrics used for evaluation.

3.1 Faster RCNN Architecture

In this project, we chose to implement the Faster RCNN Architecture [1] mentioned in section 2.2.2 It is one of the most precise detection models while also being one of the most adaptable due to its modular architecture. The model can be divided into three parts: (1) feature extraction, (2) region proposal, and (3) classification. Figure 3.1 is a representation of this architecture.

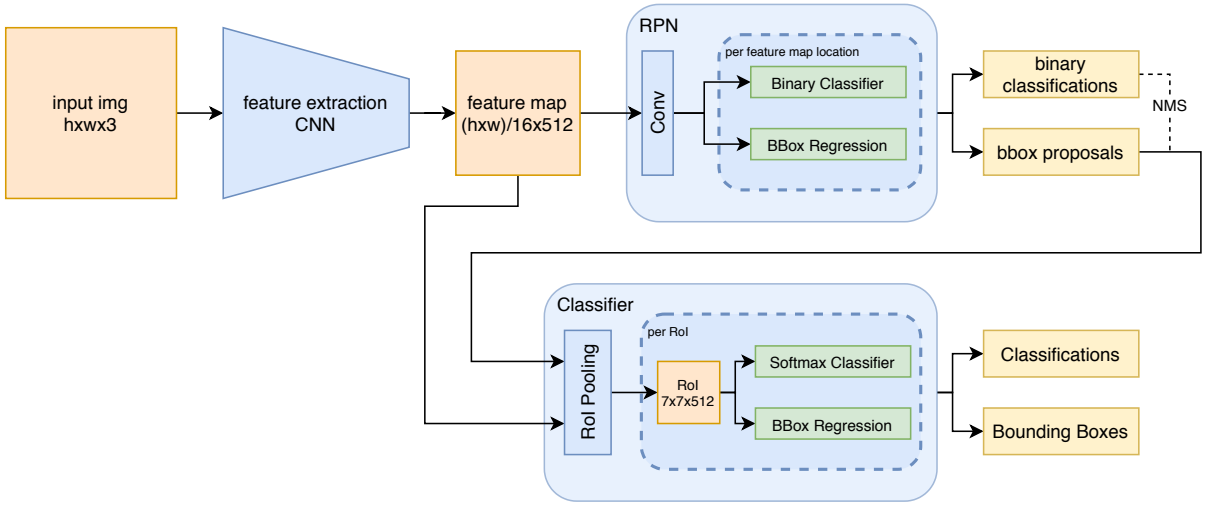


Figure 3.1: Representation of Faster RCNN architecture.

To briefly describe the architecture: an input image goes through a feature extraction CNN (VGG16 [2]), this CNN is known as the shared layers of the network. A feature map is computed and used as input to the region proposal network (RPN). This module uses a sliding window-like technique to extract a fixed number of box proposals in all image locations. The number and dimensions of the proposals at each location is also fixed. The RPN module applies two simultaneous loss functions to each box proposal: a binary classifier to attribute the probability of an object being represented by that box and a bounding box regressor which improves the proposal's location. The last section of the network, the classifier, receives as input the feature map from the CNN and the best region proposals from the RPN. Each of those proposals is extracted from the feature map and warped to a fixed size. A softmax function is used to produce class probability scores and another box regression is applied to further improve the proposal's location. The final output of the network are all these class probabilities and bounding boxes.

The network has 4 losses, two in the RPN and two in the classifier. As such, special training methods are applied, these will be further explained at the end of the section. In short, the network is not trained in one stage. The shared layers (the feature extraction CNN) are alternately trained by each of the other two modules.

The following subsections go over each of the three modules and the alternating training in greater detail.

3.1.1 Feature Extraction

This module is a basic feature extraction CNN. In the original Faster RCNN paper [1], both VGG16 [2] and ZF net [20] are implemented. Another common implementation is ResNet50 [23]. Out of the two originally implemented CNNs, VGG16 obtained better results, as such, this is the feature extraction network we will be using.

VGG networks, proposed by Simonyan and Zisserman in 2014, have one of the most uniform and simplistic architectures, they are one of the most widely used CNNs for feature extraction. VGG16 achieved great results in the ImageNet challenge ILSVRC [9] in 2014, placing second. This network is made up of 5 convolutional blocks followed by 3 fully connected classification layers. The first two blocks have 2 convolutional layers, while the remaining three have 3 convolutional layers. All of the blocks end with a max pooling layer which downsizes the feature map's spatial dimensions (height and width) to half. The main trait of this CNN is the use of very small convolution filters, every layer uses 3x3 filters.

In Faster RCNN, the VGG is slightly altered, the fully connected layers are removed, as is the last pooling layer. The network is initialized with pre-trained weights on the 1000-way ImageNet classification data, given that training VGG from scratch was deemed unnecessary [19], being better to opt for a pre-trained network and later perform fine-tuning. A representation of this modified CNN is shown in figure 3.2.

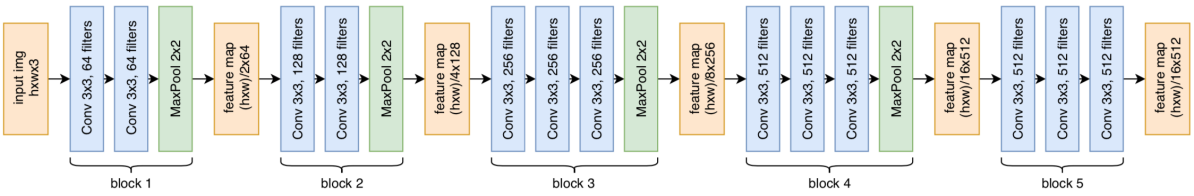


Figure 3.2: Representation of VGG16 network as used in Faster RCNN architecture.

The original VGG16 has a fixed input size of 224x224 px. Eliminating the final layers removes this restriction and this module of Faster RCNN accepts any input size. The output of this module is a feature map with height and width 16 times smaller than the input image. The receptive field of 1 output pixel is 196 pixels.

3.1.2 Region Proposal

The Region Proposal Network (RPN), takes as input the feature map computed by the CNN and returns the set of possible object locations that the classification module will use. It is comprised of one 3x3 convolutional layer and two twin 1x1 convolutional layers, which are akin to a linear transformation. After the first convolutional layer, the total receptive field is 228px. Figure 3.3 shows a representation of the RPN architecture.

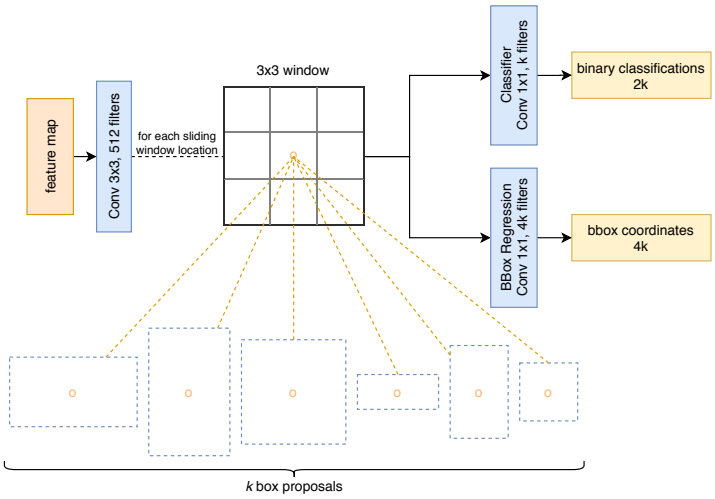


Figure 3.3: Representation of RPN from Faster RCNN architecture. k boxes of predefined dimensions are applied at the center of the 3x3 window obtained from the feature map.

The first layer is equivalent to applying a sliding window to the feature map. The convolution "extracts" a 3x3 window of the map at each location. At the center of each window, k boxes of predefined dimensions, called anchors are applied. k is an hyperparameter, it is the number of box ratios used times the number of box scales, therefore each anchor has a specific box ratio and scale. The k anchors result in k box proposals at each window location.

Each bounding box proposal is fed, simultaneously, through a binary classification layer, which says whether that region contains an object (positive) or background (negative), and a box regression layer. To perform regression on the different sized anchors, a set of *k* regressors is learned, one for each ratio and scale combination. Weights are not shared between the different regressors. The multi-task loss function used to optimize the RPN is as follows

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \tag{3.1}$$

where i is the index of a box proposal, p_i is the probability (confidence score) of it being an object, p_i^* is the GT label which is 1 if the box is positive and 0 if negative. t_i is a vector of the 4 parameterized coordinates of the box proposal and t_i^* is of a positive GT box. L_{cls} is a binary log loss and $L_{reg}(t_i, t_i^*) =$

$R(t_i - t_i^*)$ where R is the following robust L1 loss function

$$R(x) = \begin{cases} 0.5x^2, & \text{if } |x| < 1 \\ |x| - 0.5, & \text{otherwise.} \end{cases} \tag{3.2}$$

The $p_i^* L_{reg}$ ensures that this loss is only computed for positive GT boxes. N_{cls}, N_{reg} and λ are normalizing and balancing parameters. N_{cls} is the number of GT boxes in each image. N_{reg} is the number of positive GT boxes in each image multiplied by 4, which is the number of coordinates per box. In [1], it is shown that the value of λ does not have a significant influence on the results in a range between 1 and 100, as such it will be kept as 1.

The parametrization applied to the box coordinates before performing the regression is as follows,

$$t_x = (x - x_a)/w_a, \quad t_y = (y - y_a)/h_a, \quad t_w = log(w/w_a), \quad t_h = log(h/h_a) \tag{3.3}$$

$$t_x^* = (x^* - x_a)/w_a, \quad t_y^* = (y^* - y_a)/h_a, \quad t_w^* = log(w^*/w_a), \quad t_h^* = log(h^*/h_a) \tag{3.4}$$

where x y are the box's center coordinates and w h are the box's width and height. The variables x, x_a and x^* refer to the predicted box, anchor box and GT box respectively.

During training, classification is performed based on the IoU (Intersection over Union) overlap score between the annotations and the box proposals. If there is a high IoU score, then the proposal is considered positive.

At the end of this module, there are two classification confidence scores (object or not) and four box coordinates for each box proposal, of which there are *k* per anchor location. Before the box proposals are sent into the classification module, the results undergo non maximum suppression to reduce the redundancy of the proposals.

3.1.3 Classification

The classification architecture used in Faster RCNN is the same as the one implemented by Girshick in Fast RCNN [22].

It receives as input the feature map and the region proposals previously computed. For each proposal, a special type of convolutional layer, named RoI Pooling layer, extracts a fixed length feature vector from the feature map at that proposal's location. The RoI pooling layer divides every proposal into 7x7 regions, independently of the proposals original size, this is done by dividing the region's measurements, height and width, by 7, obtaining the new row and column length. These regions are known as Regions of Interest (RoI). Each RoI is transformed into a vector by two fully connected layers. The resulting vector goes through two sibling output layers, a softmax function and a per class bounding box regressor, producing class probability estimates and refined bounding box positions.

The output of the classifier module is a vector of class scores, one per class plus background, and four box coordinates, per class, for each RoI. Figure 3.4 shows a representation of this classifier.

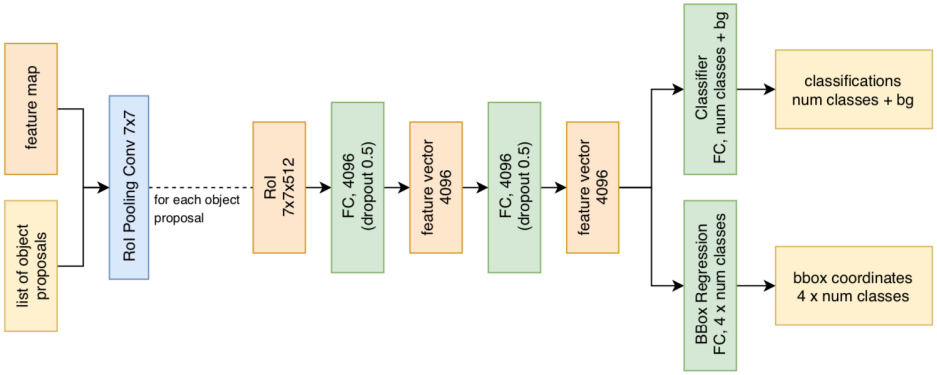


Figure 3.4: Representation of Faster RCNN classification module.

The multi-task loss used to jointly train for classification and box regression is as follows,

$$L(\{p^i\},\{t^i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p^i, p^i_*) + \lambda\frac{1}{N_{reg}}\sum_i [u \geq 1] L_{reg}(t^i_u, t^i_*) \tag{3.5}$$

where i is the index of a RoI, p^i is the class probability vector predicted for $K+1$ classes, $p^i = (p^i_0, ..., p^i_K)$, where K is the number of classes. Likewise, p^i_* is the ground-truth probability vector, in which the correct class has value 1 and all others have value 0. t^i_k, with $k \subset [1, ..., K]$, is the parameterization of the 4 coordinates of the predicted bounding box, computed for each of the K classes. t^i_* are the parameterized coordinates of the ground-truth box, which corresponds only to the true object class. These parameterizations are the same as in equation 3.3, with the a subscript representing the corresponding region proposal instead of anchor. u represents the index of the true class. L_{cls} is the categorical cross-entropy between the predicted and ground-truth class probability vectors. The bounding box regression $L_{reg}(t^i_u, t^i_*) = R(t^i_u - t^i_*)$, where R is the same smooth L1 loss as in equation 3.2. The Iverson bracket indicator function $[u \geq 1]$ ensures the term is only applicable when the true class is not background, which has index $u = 0$.

The balancing term λ is kept as 1. The normalizing parameters N_{cls} and N_{reg} are the number of RoIs and the number of positive ground-truth RoIs, respectively.

3.1.4 Training Details

Faster RCNN has a modular architecture, in which two separate parts of the network share the convolutional layers of the feature extraction network. This demands particular training techniques. Three different techniques are proposed by Ren *et al.* in [1]. In this project we will adopt the method applied in

the paper, the 4 step alternating training:

1. The RPN module is trained with an ImageNet pre trained VGG16 model. Only the altered RPN layers are saved at the end of this step, as well as the region proposals.
2. The classifier module is trained, with the same ImageNet pre trained VGG16 model as the previous step and using the regions proposed by it. Both parts of the network are trainable and their updated weights are saved.
3. The RPN module is again trained, initializing the VGG layers with those obtained in step 2 and the RPN layers with those obtained in step 1. The VGG layers, also known as shared layers, remain fixed. This step is used to fine-tune the RPN to the shared layers trained with the classifier, actually accomplishing the sharing of the feature extraction layers.
4. On the last step, the classifier layers are fine-tuned using the region proposals from the previous step and the same fixed shared layers.

To implement this training technique, we joined the 4 steps in 2 phases. Phase 1 contains steps 1 and 2, while phase 2 contains the remaining steps.

Implementation Details

We will be using as foundation a Faster RCNN implementation available at [56] on GitHub. It is a Python project using the Keras deep learning framework. This implementation is quite similar to the original one by Ren et al. in [57] written in Matlab and using the Caffe [58] framework. As backend we will use Tensorflow.

3.2 Traffic Datasets

Data is one of the main components in detection. In this thesis we will be using the CityCam dataset for training. Additionally, we will be analysing non-annotated traffic data from the city of Tallinn.

In the previous chapter, several vehicle detection datasets were mentioned. However, most of them do not coincide with our purpose. For this work, we need real-world cluttered scenes captured by traffic cameras. Preferably, footage of lower resolution as that is the norm for most traffic monitoring video systems. The two datasets that better match with these attributes are TRANCOS and CityCam. From these two, we went with CityCam as it contains many urban scenes and we consider urban traffic a very significant use-case.

CityCam

CityCam is a large labeled traffic webcam dataset created for traffic analysis by [5]. The dataset is composed of 60 million frames obtained from 212 webcams over a 4 week period. 60,000 of those

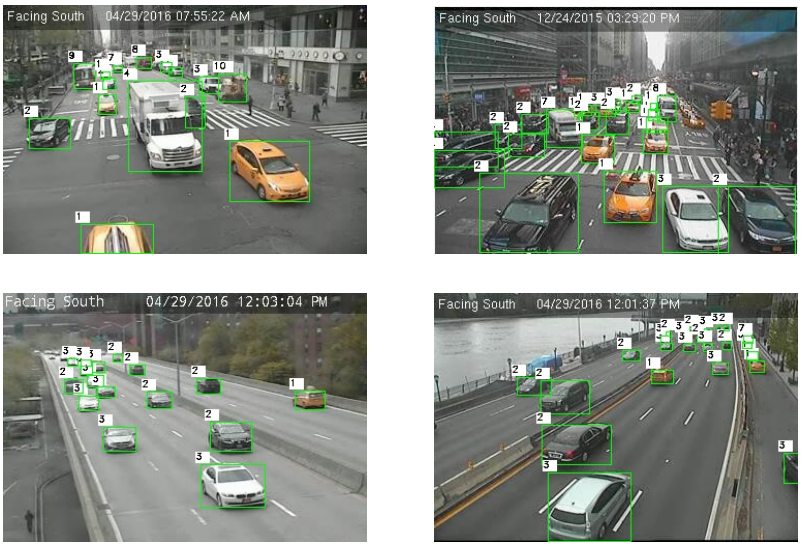


Figure 3.5: Frames from the CityCam dataset with annotations. The two top frames represent an urban traffic scene and the bottom two represent highway traffic.

frames have been annotated, containing around 900,000 labeled vehicles. It depicts real world urban traffic scenes on a large city.

All the annotated frames belong to 16 different cameras placed around New York City, 15 in Manhattan. Eleven of those cameras are placed on typical urban intersections while 5 are on higher traffic and higher speed roads, similar to highways.

The annotations provide information on the vehicle class, bounding box locations, camera orientation, the time and the weather when the footage was obtained. The vehicles in this dataset are divided into 10 classes: (1) taxi, (2) black sedan, (3) other car, (4) small truck, (5) medium truck, (6) big truck, (7) van, (8) medium bus, (9) big bus, and (10) other vehicles (such as bikes and motorbikes).

Figure 3.5 shows four frames from the CityCam dataset and the corresponding annotations: bounding boxes and vehicle class. In the next chapter we will present a more thorough analysis of the CityCam data.

Tallinn dataset

To perform further traffic analysis, we have available a dataset comprised of traffic footage from over one hundred cameras installed throughout the city of Tallinn. Example frames from this data are shown in Figure 3.6.

This data was collected during the month of March, 2020. The traffic photos were taken during the day, from 7 a.m. to 10 p.m., on average 6 photos were taken per hour.

This data is not annotated for detection, however, there is a density estimation score for each frame. The top row in figure 3.6 shows a frame (left) and its corresponding density map (right). The purpose of

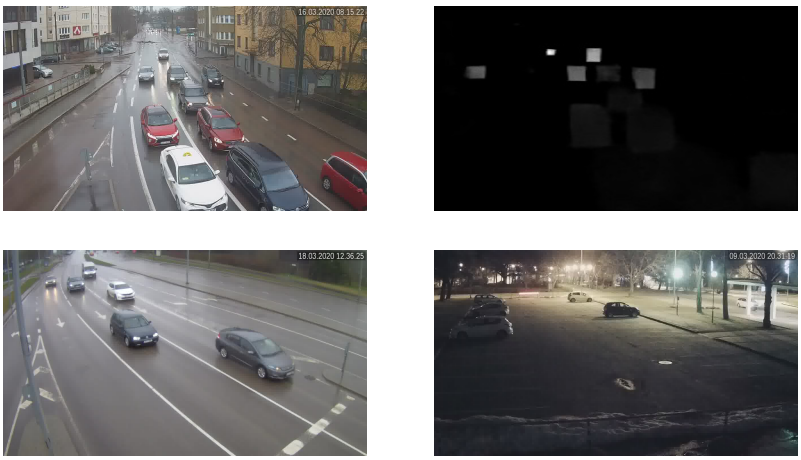


Figure 3.6: Frames from the Tallinn dataset. The top two figures show a frame from an urban scene and its corresponding density map. The bottom two figures show a highway frame and a parking lot frame.

analysing this data is to obtain temporal inferences from real traffic data, simulating a traffic monitoring system. As well as to compare both methods of vehicle counting.

3.3 Evaluation Metrics

The performance of object detection algorithms is evaluated based on three criteria: precision, recall and speed. Generally, precision and recall are represented by a single metric, mean Average Precision (mAP). There is one more measure essential to object detection, the Intersection over Union (IoU).

Intersection over Union

IoU is a measure of overlap between two bounding boxes. Having two bounding boxes, their area of overlap (intersection) and area of union are computed, they are then divided giving the Intersection over Union score (Equation 3.6). Figure 3.7 illustrates this metric on a pair of overlapping ground-truth (GT) and predicted boxes.

$$IoU = \frac{Area\,of\,Intersection}{Area\,of\,Union} \tag{3.6}$$

For evaluation purposes, a ground-truth bounding box is compared to the predicted bounding box. To determine whether a prediction is suitable, a threshold is applied to the IoU score. In most object detection benchmarks an IoU greater than 0.5 indicates the prediction is a true positive. Recently, it has become the norm to use the technique in [35] where instead of using simply one IoU value, usually set at 0.5, the results are gathered at several IoUs.

Precision and Recall

Precision is the fraction of the detections made that are correct. Recall is the fraction of correct

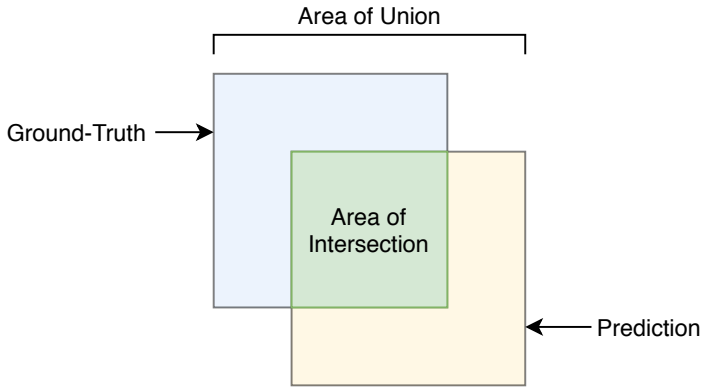


Figure 3.7: Illustration of the IoU metric.

detections made compared to the detections that should have been returned. In more general terms, precision is the number of TP (True Positives) over the number of TP + FP (False Positives) and recall is the number of TP over the number of TP + FN (False Negatives).

For each class and each image i, the detector returns predictions as (b_{ij}, s_{ij}), where b_{ij} is the predicted location and s_{ij} is the confidence of prediction j. The locations predicted are matched to ground-truth bounding boxes, using an algorithm described in [9], and binarily scored according to a certain threshold applied to the IoU overlap result, the score (z_{ij}) is 1 if the boxes IoU is greater than the threshold, and 0 otherwise. The formulas for precision and recall are as follow:

$$Precision(t) = \frac{\sum_{ij} 1[s_{ij} \geq t] z_{ij}}{\sum_{ij} 1[s_{ij} \geq t]} \tag{3.7}$$

$$Recall(t) = \frac{\sum_{ij} 1[s_{ij} \geq t] z_{ij}}{N} \tag{3.8}$$

where N is the number of ground-truth instances of a certain class, across all images.

Average Precision

Average Precision (AP) is a slightly more complex criteria which encompasses the two metrics above. For each class, AP is calculated as the average of the maximum precision over the different levels of recall, or, in other words, AP is the area under the Precision-Recall curve.

To compute the Precision-Recall curve, we begin by sorting the predictions in a descending order according to their confidence score. After, we calculate the precision and recall for the first prediction and update these values with every new prediction. As more predictions are considered, the recall score increases, ideally reaching 1. The precision scores are not as regular, they increase with TP and decrease with FP. The precision and recall values for all predictions are plotted, resulting in the Precision-Recall curve.

Before computing the area under the curve, the precision in interpolated. The precision at each recall

value becomes the maximum precision at that recall or higher (3.9). The AP is then computed as the mean of the interpolated precision at all the recall values (3.10).

$$p_{interp}(r) = \max_{\tilde{r}:\tilde{r}\geq r}(p(\tilde{r})) \tag{3.9}$$

$$AP = \int_0^1 p_{interp}(r)dr \tag{3.10}$$

mean Average Precision

The mean Average Precision (mAP) is simply the average of the AP over all classes. Most object detection benchmarks use mAP to evalute results, however, some compute this metric according to certain criteria. In [35], they present mAP across three scales: small objects ($area < 32^2$), medium objects ($32^2 < area < 96^2$) and large objects ($area > 96^2$). In [46], they compute the mAP at three difficulty levels, using the bounding box height, occlusion level and truncation percentage as criteria. In terms of minimum box height the levels are: easy ($40px$), medium ($25px$) and hard ($25px$).

In our evaluation we will also be using three levels of mAP according to bounding box dimension. The box dimension values are adapted to the CityCam dataset and as follows:

- small: width $\leq$ 13px;
- medium: 13px $<$ width $\leq$ 29px;
- large: width $>$ 29px.

These values are explained in the next chapter, 4.1. From these values, the difficulty levels are: (1) easy - large boxes; (2) medium - medium and large boxes; (3) hard - small, medium and large boxes. Because the dataset does not include annotations relative to the occlusion and truncation level of the objects, we cannot use these as criteria.

Detection Speed

Detection speed is measured in Frames per Second (FPS), *i.e.* how many images the detector can evaluate in one second. The constraints to this measure depend immensely on the application. Vehicle counting is usually employed as an offline method. This implies that time constraints are not too severe.

4

CityCam: Analysis and Training

Contents

The most significant part of our work is the adaptation of a general object detection architecture, Faster RCNN, to the low-resolution traffic dataset CityCam. To achieve this we begin by analysing the data in the first section. In the second section, we adapt the network's parameters with the results obtained by studying CityCam.

4.1 Analysis of the CityCam Dataset

In this section we analyse the CityCam dataset. The information given is on the following topics: data format, scene type, annotation box statistics, box scales and ratios, class distribution and finally, weather conditions.

The CityCam dataset is available as video files and .xml annotation files. The video files are divided by camera and date, containing a varying number of frames, between 100 and 564, with an average of 300 frames per video. The majority of the videos (except on 3 of the 16 cameras) present a frame rate of around 1 frame/second. All the frames have a resolution of 240x352 pixels.

For this implementation it is beneficial to first convert the videos to individual .jpg frames and the annotations from a per frame .xml file to a general .txt file. Each line in this annotations file represents one vehicle instance in the following format: path to the frame, followed by the coordinates to its bounding box (x1, y1, x2, y2) and finally, the vehicle type id (i.e. the object's class).

The full dataset was partitioned as subsets of 90% for training+validation and of 10% for testing. The training+validation set was further partitioned into 80% for the training set and 20% for validation.

Scene Type

The CityCam dataset is composed of images from 16 different cameras, most of these are installed in urban intersections, depicting regular urban traffic. Observing the footage from each camera, we have decided to divide them into three categories: urban, highway and other. The distribution of scene type in the CityCam training set is as follows: urban - 78.26%, highway - 16.24%, other - 5.5%. In table 4.1, the camera id's are colour coded: urban - green, highway - blue, other - red.

Figure 4.1 shows a typical frame from each of the cameras. The *other* category contains two cameras that depict uncommon urban traffic: 164 and 166 (the first frames in figure 4.1. The highway cameras have the id's 253, 572 and 691. All other cameras show basic urban traffic.

Initial Analysis of Annotations

For the subsequent parts of the implementation it is relevant to have an understanding of the dimensions of the objects present in the data, especially in terms of bounding box scales and box ratios. Table 4.1 contains the median values of the areas, widths and heights of all bounding boxes per camera, as well as other statistics.

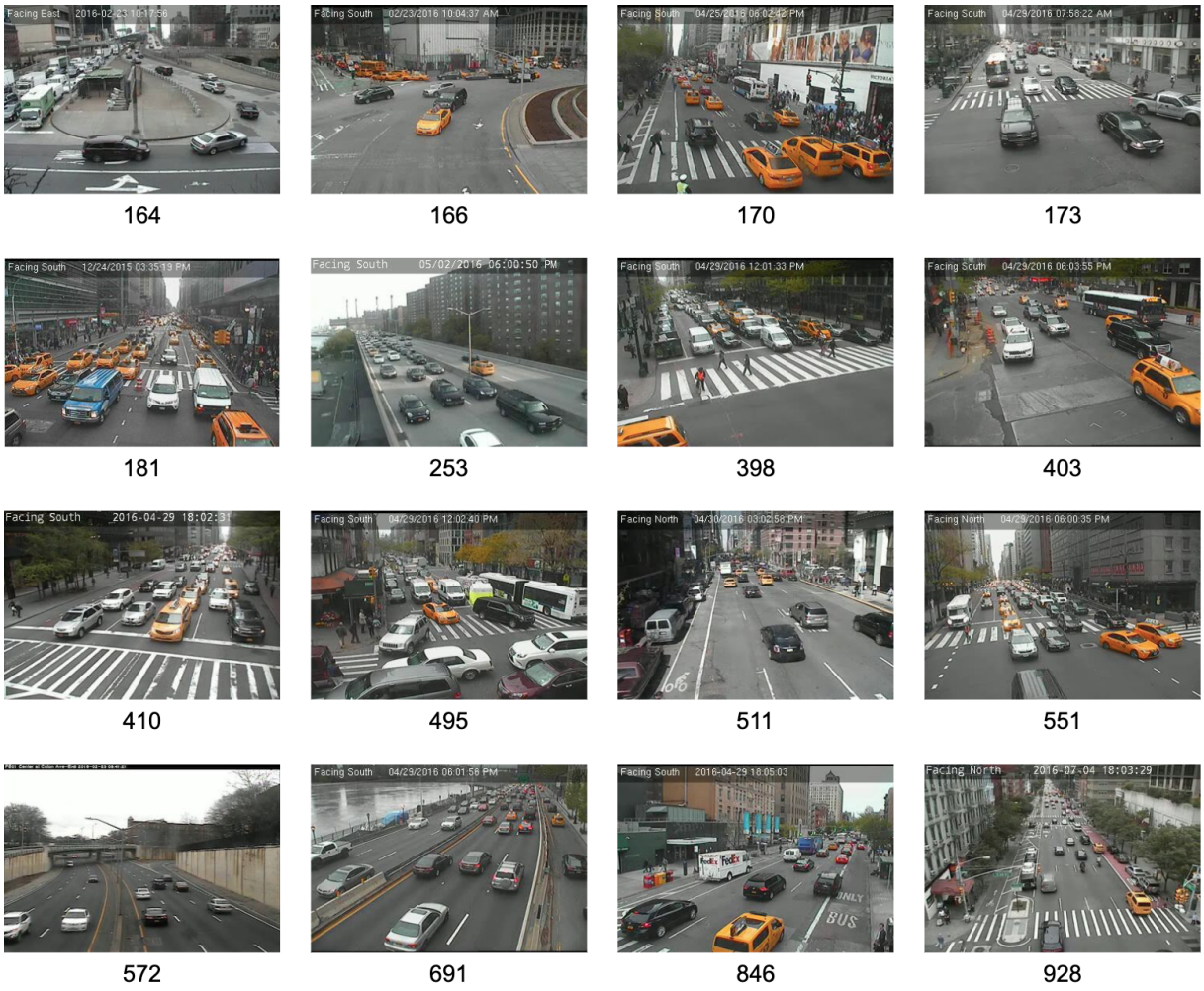


Figure 4.1: Example of CityCam frames per camera. The number under each image is the camera id.

With these values we can confirm two expected characteristics from this type of traffic footage. First, each frame tends to contain a very large number of annotations. The average over the entire training set is of 14 vehicles per image. Second, the size of the annotations is extremely small for common object detection applications. For example, in the MSCOCO challenge [35], annotations with an area inferior to $32^2 = 1024px$ are classified as *small* and in the KITTI [46] traffic dataset, only annotations with a height greater than $25px$ are considered for evaluation. In CityCam, at the original image size, the average width and height don't even reach $25px$.

By observing the values in the table, 4.1, we can also see that the three highway cameras (id in blue) present considerably smaller median dimensions than the rest of the dataset. This could imply that the detector's performance will be lower on the highway scenes of CityCam.

Bounding Box Scales and Ratios

One of the challenges in vehicle detection is the discrepancy in object scales. The objects closest

Table 4.1: Analysis of CityCam training set. The table shows the distribution of frames and annotations per camera, as well as the median of annotations in each image. The last three columns present the median area, width and height per camera and the average over all cameras. The colours in the camera id column represent the scene type: urban-green, highway-blue, other-red.

	Num Frames		Num Annotations per camera		Num Annotations per frame	Median Annotation Dims		
Cam id	count	%	count	%	median	area	width	height
164	1098	2.11	26484	3.68	23	380	23	16
166	1732	3.39	20357	2.85	12	396	29	14
170	4996	9.79	34612	4.8	6	500	24	20
173	4466	8.75	42132	5.88	9	399	22	19
181	450	0.88	8292	1.15	19	361	21	18
253	3184	6.24	57454	7.97	17	121	13	9
398	4672	9.16	83508	11.63	18	336	22	15
403	3504	6.87	47398	6.61	13	532	27	19
410	4836	9.48	72151	10.05	15	462	23	20
495	4291	8.41	60892	8.49	14	306	21	15
511	4733	9.28	46290	6.44	9	208	17	13
551	3176	6.22	59426	8.28	18	240	17	14
572	1876	3.68	17058	2.37	8	190	16	12
691	3224	6.32	78118	10.87	24	117	12	9
846	2404	4.71	16868	2.35	7	400	22	17
928	2407	4.72	46907	6.57	18	165	14	12
Total	51049	100%	717947	100%	-	-	-	-
Avg	-	-	-	-	14.38	319.56	20.19	15.13

to the camera are much larger than the ones farther away. To study this in our dataset, we performed a clustering of the widths and the heights of all the annotations in the training+validation set. Our main goal with this clustering is to obtain three box scales and three width/height ratios that are representative of the CityCam data and that can be used during the training of the Faster-RCNN detector, given that this architecture requires the specification of these parameters in the RPN stage. We used Kmeans to cluster each of the measures into three groups with the intention of obtaining three scales: small, medium and large.

Table 4.2 shows the results of the analysis used to obtain the scale values. For each of the two measures, width and height, we computed three clusters per camera. Each of the clusterings was done with all the training+validation set bounding boxes of that camera. We obtained three width clusters and three height clusters per camera and their respective centroids. Each cluster represents a scale type: small, medium or large.

To obtain the scale values, we chose to use the width clusters' centroids. For each of the scale types, we averaged the corresponding centroids of all cameras. This average was performed taking into account the weight of each camera in the dataset and the size of cluster. This resulted in three bounding box widths that represent the scale values to be used further on. The results attained from this analysis are 13, 29 and 65.

Table 4.2: Results of Kmeans algorithm applied to the widths and heights of all bounding boxes in the training+validation dataset with K=3. Colored boxes indicate the largest width and height cluster of each camera.

	Width clusters						Height clusters					
	S		M		L		S		M		L	
Cam id	value	%	value	%	value	%	value	%	value	%	value	%
164	15	51.6	31	33.6	89	14.8	10	43.7	19	32.3	37	24
166	14	34.4	31	40.8	55	24.7	10	54.3	22	31	46	14.6
170	14	54.7	42	30.7	104	14.6	10	40.9	24	34.7	50	24.4
173	14	46.3	28	36.6	84	17.1	12	57	30	29.8	58	13.2
181	15	58.7	33	35.1	95.5	6.2	12	48.9	23	37	47	14.1
253	9	57.3	22	28.3	50	14.4	7	65	17	24.5	35	10.5
398	16	59.1	35	33.1	75	7.8	11	57	24	34.5	53	8.6
403	19	55.1	39	36.9	95	8	13	50.2	26	41.4	64	8.3
410	15	50.5	33	30.9	60	18.6	11	42.8	25	32.2	48	25
495	15	60.7	33	33.9	89	5.4	11	52.1	20	35.9	44	11.9
511	13	62.3	29	23.9	61	13.8	10	64.2	24	24.3	53	11.5
551	12	52.3	23	37.2	65	10.5	10	54.3	20	34.8	44	10.9
572	10	50	21	35.2	43	14.8	7	48.4	16	34.4	30	17.2
691	9	61.6	23	26.7	55	11.6	7	64	18	25.3	40	10.8
846	14	54.8	41	25.5	75	19.7	9	44	21	30.4	35	25.5
928	9	43.2	16	38	31	18.8	9	58	18	29.5	30	12.5

To obtain the ratio values, we performed another clustering analysis, based on the assignments of the previously described Kmeans. Taking the largest clusters from each camera, the instances are again clustered into three groups. Resulting in three width clusters and three height clusters which are combined to produce the most representative ratios between width and height. These are [0.8 0.6], [1.1 0.8] and [1.4 1.1], where the first number in brackets is for the width and the second for the height.

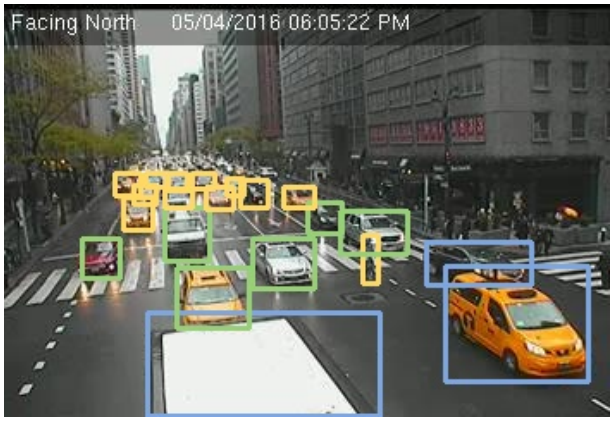


Figure 4.2: Example of the assignment of each annotation to a cluster in a frame of CityCam. Each box color represents a cluster: yellow belongs to the small cluster, green to the medium and blue to the large.

In Figure 4.2 we show an example of how the annotations in a frame are assigned to one of the three clusters, based on the scale and ratio values defined above.

Class Distribution

Table 4.3: Class counts in training+validation set. The table contains the class distributions for the total set and for each of the three types of scene. Values above 15% are shown in bold font.

class id	1 taxi	2 black sedan	3 other car	4 small truck	5 medium truck	6 big truck	7 van	8 medium bus	9 big bus	10 other	total
total count (x1000)	113	240	259	12	19	4	27	7	17	19	718
total %	**15.72**	**33.48**	**36.05**	1.72	2.7	0.61	3.7	1.04	2.32	2.66	100%
urban %	**19.1**	**30.8**	**32.76**	1.81	3.04	0.49	4.03	1.26	3.12	3.61	100%
highway %	4.99	**44.27**	**46.76**	1.22	0.35	0.03	1.85	0.29	0.03	0.19	100%
other %	13.28	**28.04**	**37.55**	2.35	6.57	3.82	6.04	1.05	1.01	0.28	100%

CityCam is a dataset representative of real traffic conditions, as such, it is expected to have an imbalanced class distribution. When looking at a traffic scene, it is clear that some types of vehicle are much more common that others, and this is confirmed by performing class counts on the training+validation data, presented in Table 4.3. The three first classes solely constitute over 85% of the entire dataset. These are the vehicle subclasses that represent the *car* class. All other vehicle types occur a noticeably smaller number of times, which might become an obstacle to the correct classification of instances belonging to these classes.

Comparing the class distributions between the different traffic scenes and the total set, we can note that the urban cameras present the most similar distribution to the total set one. This is expected given that most of the cameras are installed in typical urban roads. We also see that the *highway* set is almost entirely composed of black sedans and other cars (91%) and the *other* set has considerably more trucks than the rest. Table 4.4 shows the class distribution per camera.

Weather Conditions

The annotations of CityCam include a weather id for each of the frames. These belong to three categories: sunny, cloudy or rainy. In almost all cases, the weather condition (and the id) remains the same throughout an entire video. Although useful, these labels present no indication as to how severe the weather condition is in each of the videos.

By manually observing each of the videos, we divided them into four weather categories: normal (151 videos), rain (23 videos), heavy rain (5 videos), shadows (21 videos). In figure 4.3 we can see examples of each of the categories.

It is clear that the frames classified as *heavy rain* present a much more severe distortion. It is unlikely that an object detector will perform acceptably on these images.

This dataset contains a few errors in its annotations, mostly by having extremely small areas that don't represent any vehicle. These are unlikely to have any effect on the detection algorithm, due to the

Table 4.4: Class distribution per camera in CityCam train+val set. The colours in the camera id column represent the scene type: urban-green, highway-blue, other-red. Numbers above 15% are shown in bold font.

cam \ class	1 taxi	2 black sedan	3 other car	4 small truck	5 medium truck	6 big truck	7 van	8 medium bus	9 big bus	10 other
164	5.89	**30.61**	**38.8**	3.55	8.74	4.95	6.31	0.46	0.55	0.11
166	**22.9**	**24.7**	**35.91**	0.78	3.73	2.35	5.68	1.82	1.62	0.51
170	**31.81**	**21.26**	**16.21**	1.08	1.71	0.56	2.52	3.98	12.86	8.01
173	**20.78**	**29.26**	**25.63**	1.33	2.4	0.4	3.75	2.43	6.18	7.83
181	**45.22**	**23.02**	**16.43**	0.13	0.65	0.05	3.51	3.85	5.03	2.11
253	5.33	**48.21**	**42.55**	1.29	0.22	0.01	1.89	0.28	0.05	0.17
398	**16.29**	**27.8**	**40.72**	1.49	3.49	0.36	4.17	1.05	2.33	2.29
403	**17.8**	**31.23**	**32.02**	3.5	3.24	0.92	3.83	1.61	2.61	3.24
410	**17.28**	**36.79**	**30.21**	1.98	5.26	0.51	4.63	0.62	0.9	1.81
495	12.81	**34**	**39.91**	1.43	4.2	0.3	2.57	0.44	1.18	3.17
511	**22.33**	**23.49**	**35.15**	2.91	2.03	0.14	6.28	0.91	1.88	4.89
551	**20.5**	**32.3**	**32.89**	1.89	2.08	0.33	3.78	0.64	2.32	3.27
572	1.01	**31.71**	**60.68**	1.65	2.2	0.23	1.83	0.56	0.1	0.02
691	5.61	**44.12**	**46.82**	1.08	0.05	0	1.83	0.24	0	0.25
846	13.55	**39.22**	**23.03**	1.02	3.06	1.14	9.44	0.41	5.09	4.04
928	**17.88**	**34.42**	**36.44**	1.23	1.26	0.95	2.53	1.22	2.21	1.86

(a) rain **(b)** heavy rain **(c)** shadow

Figure 4.3: Examples of Citycam frames affected by different weather conditions.

way in which the training data is generated, explained at the end of the next section, 4.2. Also, these errors don't occur enough to be significant.

4.2 Network Parameters for CityCam

The Faster RCNN architecture was created having general object detection tasks in mind. Considering the main differences between this type of task and vehicle counting, it is likely that adapting the parameters to the CityCam traffic data will improve the detection results. In this section, we will present three of the most relevant network parameters and how we adapt them to our data: input size, anchor scales and ratios, IoU thresholds.

Input Image Size

One of the most significant parameters in this architecture is the image input size. This affects the dimensions of the feature map produced by the Feature Extraction module (the VGG) which will, in turn, affect the number of anchor locations to be analysed. The VGG used, outputs a feature map with dimensions 16 times smaller than the input's.

Assuming the dimensions of the CityCam data, 352x240px, the VGG produces a feature map with a size of 22x15. As mentioned in section 3.1.2, the RPN uses a fixed-size sliding window to extract a set number of anchors from each feature map location. With a feature map of 22x15 we have 330 locations and each represents a 16x16 section of the original image. We are using nine anchor boxes per feature map location, so this results in 330x9=2970 anchors per image. In table 4.5, we show how different rescalings of the input image alter these details.

In the original implementation, an input image is resized so that its shorter side is equal to 600px, and the larger side is resized according to the initial ratio, up to a maximum of 1000px. Assuming the dimensions of the CityCam data, 352x240px, this is a scaling 2.5 times larger.

Table 4.5: Effect of different image scalings on the region proposal stage. This table presents network details using five possible scalings of CityCam frames, from 1x to 3x. The feature map size is 16x smaller than the image size. Using a sliding window, anchors are applied to each feature map location. The location's size and area are shown relative to the original image size. The number of anchors is calculated assuming nine anchors per window location.

Scale	1x (original)	1.5x	2x	2.5x	3x
Input img size	352x240	528x360	704x480	880x600	1056x720
Feature map size	22x15	33x22.5	44x30	55x37.5	66x45
Num window locations	330	726	1320	2035	2970
Location size	16x16	10.7x10.7	8x8	6.4x6.4	5.3x5.3
Location area	256	114	64	41	28
Num anchors (k=9)	2970	6534	11880	18315	26730

By performing a scaling of 2.5 times, i.e., equivalent to the parameters in [1], the network has 6.25 times more anchor locations to evaluate, which will have a considerable impact on the amount of time the network spends per image. However, while expanding the image size decreases the time efficiency, it is practically a necessity in the CityCam dataset.

In Table 4.1, we can observe that over a third of the cameras have annotations with a median area smaller than 256px = 16x16. This means that, when not performing a scaling of the image, half of the annotations' areas are smaller than the windows considered, and consequently not captured by the network. When using an input two times larger that the image, the sliding window area corresponds to a 64px = 8x8 area in the original image, which is an area small enough to capture most of the vehicles in the dataset.

Anchor Scales and Ratios

In the RPN module, Faster RCNN uses fixed-size anchor boxes to extract region proposals from each section of the feature map. During training, these proposals are considered positive only when they have an IoU higher than a threshold with a ground-truth (GT) box. For this to be the case, the dimensions of the anchors have to be representative of the GT object's dimensions.

In the original implementation, the anchor scales and ratios were defined to possibly represent large objects from diverse classes. They applied nine bounding box proposals per feature map location. These are combinations of the scales 128, 256, 512 with the ratios 1:1, 2:1, 1:2. We will be comparing these to the values computed for our data in the previous section (4.1): [13, 29, 65] and [0.8:0.6, 1.1:0.8 , 1.4:1.1] for the scales and ratios, respectively.

The scale dimensions refer to the input image size and not the feature map. Also, the first element of the ratio is applied to width while the second to the height. Table 4.5 shows the maximum number of anchors produced according to the image input size when using k=9 anchors per location. The values presented in the table are only approximate given that the network dismisses proposals that exceed the image boundaries.

It is evident by these original values, that their intention is to find large objects that occupy a considerable part of the image, the smallest proposal area is 128x128px. This is the usual purpose of image detection challenges, however, it is not applicable to our dataset which contains several small objects as opposed to a main larger one.

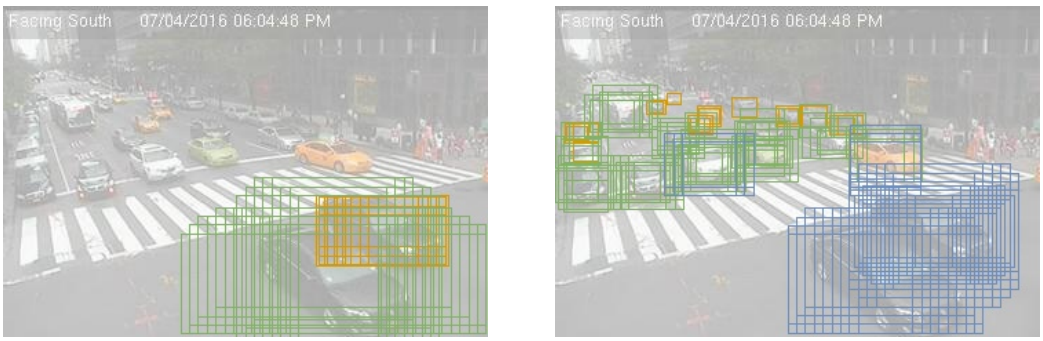
Figure 4.4 shows an example of all the positive object proposals that would be generated for one image. On the left with the original parameters and on the right with the values obtained with the clustering analysis of the dataset shown in the previous section (4.1). In both examples we used as input an image 2.5x larger than the original CityCam frame size. Each color represents an anchor scale: small - orange, medium - green, large - blue.

On the first image 4.4(a), only two vehicles are encompassed by proposal boxes, one with the small scale, and one with the medium. In contrast, on the right image 4.4(b), only three of the 24 vehicles have no proposal associated, and all box scales are present. This serves to show the impact the correct definition of this parameters will have in the execution.

IoU thresholds

The IoU thresholds are parameters applied during the RPN training. IoU overlap is the main metric to assess how appropriate a box proposal is to a ground-truth bounding box. In RPN, these thresholds which determine whether a box proposal is positive or negative, i.e., if it contains an object or not. To do this, two thresholds are used, and upper and a lower one. The following are the criteria used by the network:

- Positive proposal:



(a) Original anchor dimensions

(b) Anchor dimensions computed for CityCam

Figure 4.4: Example of positive anchor proposals in a CityCam frame.

  - highest IoU score with a GT box
  - IoU $>$ upper threshold with any GT box

- Negative proposal:
  - IoU $<$ lower threshold with all GT boxes

All other boxes are irrelevant during training and are thus eliminated.

In the original implementation, the values used are 0.7 and 0.3, for the upper and lower threshold respectively. These values are appropriate to strictly define and ”teach” the network what a positive region is versus a negative one. However, they are slightly too restrictive when applied to this traffic data. An analysis on a sample of the dataset (around 10%), showed that using the anchor dimensions previously defined in this section, the average IoU is of around 0.55. As such, it is more appropriate for this data to use smaller thresholds. Such as 0.5 and 0.1, for the upper and lower threshold respectively. The lower threshold is altered so that the difference between positive and negative proposal is maintained.

At the end of the RPN module, non-maximum suppression is applied using another IoU threshold. If two proposals have an IoU overlap higher than this value, then only the one with the highest class confidence score is kept. This comparison is done to all the positive and negative proposals, resulting in a fixed number of proposals per image that advance to the classification module. The IoU threshold is set at 0.7, as in the original implementation, and is left as such during this project.

Generation of Training Data

The training of Faster RCNN follows a particular sampling technique, the ”image-centric” strategy from [22], that is, the mini batches used are of only one image. For this to be possible, each image requires a ground truth composed of a high number of positive and negative regions, ideally a total of 256 regions, 128 each. For this reason, we can’t directly use the dataset’s bounding boxes for train-

ing. Instead, Faster RCNN generates its own ground truth (GT) from each image's annotations. This generated GT should follow the anchor dimensions later used in the RPN module.

The generation of this GT is very similar to the anchor box generation of the RPN, in section 3.1.2. The steps are as follows:

1. rescale input image to the dimensions of the feature map;
2. use a sliding window to go through all image locations;
3. apply bounding boxes at each location, according to RPN anchor dimensions;
4. compute each box's IoU overlap with all image GT annotations;
5. classify each box as positive, negative or ignored according to the IoU score:
    - positive: IoU $>$ upper threshold OR highest IoU of a certain GT annotation;
    - negative: IoU $<$ lower threshold with all GT annotations;
    - ignored: all other boxes.

While a positive box is only associated to one GT annotation, there is no limit to how many generated proposals can represent the same GT box. Given the small chance that no generated box overlaps, by any amount, with a GT annotation, that annotation is ignored. The IoU thresholds applied in this process should be the same as the ones used in the RPN.

For each training image, the aim is to extract 256 region samples for the network to use as ground truth. Ideally, 128 positive and 128 negative. This is, however, a difficult proportion to attain with our traffic data. If the number of generated boxes labeled as positive is inferior to 128, all the positive boxes will be used and the remaining of the 256 samples are randomly selected negative boxes.

We analysed the generation of ground-truth on a sample of the CityCam dataset. We used a representative sample of 4000 images and applied the generation algorithm explained above. We tested using two different bounding box scales and ratios combinations, the ones from [1] and the ones computed for our traffic data. We also used three input image scales: 1x, 2x and 2.5x. The IoU thresholds used are 0.1 and 0.5 for the lower and upper threshold, respectively. Table 4.6 shows the results obtained.

For each image in the dataset sample used, we generated all boxes and categorised them according to the IoU scores with the GT annotations. In the table we show the average number of positive, negative and ignored boxes generated. Column *num true pos* has the number of positives obtained considering the first parameter (IoU $>$ 0.5) while column *num total pos* also contains the positives obtained with the second parameter (highest IoU of a certain GT annotation), that is, the boxes labeled as positive for lack of genuine positive instances with a certain GT annotation. The table also has the mean IoU between the positive boxes and their corresponding GT box. The final column presents the percentage of positive

Table 4.6: Simulation of the generation of training data in a sample of 4000 CityCam images. The table contains the average number of positive, negative and ignored boxes generated according to box dimension and input image scale. It also contains the mean IoU of positive boxes and the percentage of positive boxes a sample of 256 regions per image would contain.

bbox dimensions	input scale	num true pos	num total pos	num neg	num ignored	mean iou	% pos in 256 regions
ours	2.5x	95	98	12294	2078	0.573	38
	2x	63	67	7990	1354	0.569	26
	1x	15	24	1935	333	0.486	9
original	2.5x	49	61	4740	1903	0.371	24
	2x	23	36	2270	994	0.32	14
	1x	1	15	165	70	0.127	9

instances that would be used in a sample of 256 regions, as mentioned, an ideal percentage would be of 50% but that is extremely difficult considering the size of the CityCam annotations.

As expected, the best results arise when using the highest input size and the bounding box dimensions computed specifically for the dataset.

In the previous section (4.1), we mentioned that the dataset contains a small amount of errors. This method of generating training data ignores most of the extremely small GT annotations. This occurs because after dividing the dimensions by 16, an already small area becomes almost zero and is rounded to zero, having no IoU overlap with any generated box.

5

Results

Contents

This chapter presents the results achieved by training and testing our adapted Faster RCNN model on the CityCam dataset and testing on the Tallinn traffic data.

5.1 Training on CityCam

The training on CityCam was performed in two stages: initial parameter testing on a single camera and analysis of results on the full dataset.

The training of an object detector is a very time intensive task. In the Faster RCNN architecture, the processes of generating data and applying anchor boxes have a great impact on the training speed of the system. For this reason, the initial parameter testing was done on only one camera as to minimize the amount of training time spent on the entire dataset. The first subsection goes over these results.

In the second subsection, the best model tested on a single camera is trained on the full CityCam dataset. The classification and detection results are presented. In this section the effects of weather conditions are also studied and we show the results of a model trained exclusively on highway data. The final part of the section compares the results of our model to state-of-the-art object detectors.

5.1.1 Single Camera

The initial testing was performed on one camera. The chosen camera (Cam id = 398) is the one with the highest number of annotations and it's taken from a typical urban intersection. Around 4000 frames from this camera were used for training, the class distribution resembles the total one (in 4.3) and the median dimensions of the annotations are similar to the total average.

To test and perform parameter adaptation we trained several models, introducing only one change in each new one. Table 5.1 contains each models' parameters. Every new parameter was changed according to the best previous model. The models were trained from an Imagenet pre-initialised VGG16 and following the technique in Section 3.1.4 for 8 epochs per phase. As preprocessing, the images are normalised and the mean pixel per depth channel is subtracted.

Table 5.1: Parameter changes on Faster RCNN models.

model id	anchor scale and ratio	input img size	iou	num rois	optimizer	learning rate	weigth decay	amsgrad
base	og	600 px	og	300	sgd	1e-4	5e-4	-
a	our	240 px	og	300	sgd	1e-4	5e-4	-
b	our	240 px	og	300	adam	1e-5	0	FALSE
c	our	240 px	our	300	adam	1e-5	0	FALSE
d	our	240 px	our	300	adam	1e-5	0	TRUE
e	our	240 px	our	300	adam	1e-5	1e-5	TRUE
f	our	240 px	our	2000	adam	1e-5	1e-5	TRUE
g	our	480 px	our	2000	adam	1e-5	1e-5	TRUE

The next Tables, 5.2 and 5.3, contain the testing results of each model in the detection task and classification task, respectively. Throughout this section, the results are presented according to the difficulty levels previously defined: hard (s+m+l), medium (m+l) and easy (l), based on the three annotation sizes: small (s), medium (m) and large (l).

Table 5.2: Detection results using different Faster RCNN parameters. The table shows the mean Average Precisions (mAP) and mean Recall (mRec) at the three difficulty levels defined in Section 3.3. The last column contains the detection rate in seconds per image. The best value obtained in each metric is in bold.

	mAP (%)			mRec (%)			Rate
model	s+m+l	m+l	l	s+m+l	m+l	l	(sec/img)
base	15.19	22.25	43.47	19.31	28.28	55.25	0.7
a	13.64	18.81	37.98	25.34	34.95	70.57	0.4
b	34.82	45.07	68.11	45.97	59.5	89.93	0.2
c	42.29	54.04	73.5	54.2	69.26	94.2	0.2
d	43.31	55.38	76.42	53.55	68.48	94.5	0.2
e	43.25	55.34	76.47	53.17	68.02	94	0.2
f	44.07	56.15	74.05	56.62	72.15	95.15	0.2
g	**79.96**	**88.32**	**90.87**	**87.73**	**96.9**	**99.7**	0.5

Table 5.3: Classification results using different Faster RCNN parameters. The table shows the mean Average Precisions (mAP) at the three difficulty levels, the Average Precision (AP) of the three main classes (taxi, black sedan, other car) at the medium level and the mean Recall (mRec) at hard and medium difficulty. The best value obtained in each metric is in bold.

	mAP (%)			AP (m+l) (%)				mRec (%)	
model	s+m+l	m+l	l	taxi	black sedan	other car	top 3 cls	s+m+l	m+l
base	5.9	8.51	16.71	33.67	19.13	17.37	21.08	11.05	23.91
a	11.48	15.77	33.9	21.25	19.89	13.16	16.99	19.31	26.82
b	34.04	43.69	68.8	52.65	43.73	34.24	40.98	41.89	54.83
c	42.55	53.63	74.51	64.48	54.84	41.58	50.45	51.82	66.32
d	42.47	53.65	76.12	65.46	53.32	43.65	51.1	51.21	65.42
e	42.22	53.34	75.9	62.43	53.06	44.2	50.69	50.89	64.94
f	43.44	54.7	73.61	65.41	53.36	44.39	51.45	54.36	69.35
g	**76.85**	**86.62**	**90.63**	**89.95**	**86.24**	**85.17**	**86.45**	**83.93**	**94.67**

The first model tested has all the original Faster-RCNN parameters. As we can see in Tables 5.2 and 5.3 it provided very poor results. As expected, barely any correct detections were made due to the size and scale of the anchor box proposals. These dimensions were specified for general object detection, where the focus is on a few large objects. Unlike traffic vehicle detection, where images contain a large number of vehicles at different scales.

This is the only model trained with an input image height of 600px (shortest image side). The size of the input affects tremendously the model's training time, however, in this architecture, it has a great positive effect on the results. As explained in Section 4.2, the input image size dictates the number of anchor box locations and sizes: a larger image implies more and smaller locations. For the purpose of tuning the parameters, we trained most models without resizing the images (height = 240px), only on

the last model did we upscale them to twice the size (height = 480px) to compare the results.

The first change introduced (model *a*), besides the input size, was to the anchor box ratios and scales, changing them to the values calculated in Section 4.2. The results between the base model and model *a* were similar according to Tables 5.2 and 5.3. However, had we increased the input image size in model *a*, the values would be significantly higher. As such, we consider the change in anchor scales and ratios to our parameters an improvement.

Changing the optimizer to Adam greatly increased the results, as well as using the IoU thresholds defined by us in Section 4.2. Other parameters introduced minor changes. As we expected, the greatest improvement resulted with the upscaling of the input image in model *g*. The last column on Table 5.2 shows the testing time per image in seconds. We can see an increase in testing time on the last model. This is also an effect of the larger image input size. With this, we can conclude there is an obvious trade-off between detection accuracy and time. In this traffic monitoring application, a frame rate of two images per second is entirely acceptable so it is preferable to have higher accuracy and lower frame rate.

As mentioned, all models have a pre-initalised VGG as the feature extraction model as this is the standard practice [19]. We also experimented in training a model from a randomly initialised VGG, using the He normal initialiser [59]. All other parameters of this model are identical to model *e* and comparing both, we obtained better results on model *e*. The mAP in the classification task, at medium difficulty, went down from 53% to 48% and in the detection task, it went from 55% to 51%.

With this analysis on only one camera, we concluded that our parameter definition in the previous chapter (Section 4.2) positively affects the performance of Faster RCNN on the CityCam traffic dataset. Having achieved these results, we proceeded to train a model with the parameters equal to model *g* on the full CityCam training set.

5.1.2 Full Training set

After optimizing and testing on a single camera, the network was trained simultaneously on all 16 cameras using the parameters of model *g* in Table 5.1. As before, the frame rate during testing is around 2 frames per second.

Detection Results

In Table 5.4 we show the results for detection only. Below the individual camera results we present the average over all 16 cameras and the averages per scene category: urban, highway and other. In bold font, we highlight what we consider the most relevant metric on the Table: the mAP at the medium difficulty level, that is, ignoring the extremely small annotations (width < 13).

Observing the table, we can note that, as expected, the best values overall are obtained on urban scenes. This occurs because most of our training data portrays this type of traffic scene, as such, the

Table 5.4: Detection results on all CityCam cameras. The table shows the mean Average Precisions (mAP) and mean Recall (mRec) at the three difficulty levels defined in Section 3.3. The colours in the first column indicate the traffic scene type and the final rows show the average per scene type.

	mAP (%)			mRec (%)		
	s+m+l	m+l	l	s+m+l	m+l	l
164	68.64	**76.55**	81.5	82.06	91.52	97.44
166	71.33	**76.94**	82.16	84.55	91.2	97.39
170	71.41	**79.5**	82.91	84.43	93.98	98.01
173	75.47	**83.03**	88.32	84.28	92.71	98.63
181	76.7	**85.22**	88.29	85.73	95.25	98.68
253	54.59	**73.81**	78.08	69.11	93.44	98.85
398	78.7	**85.49**	90.01	87.04	94.54	99.55
403	83.58	**87.03**	90.84	91.1	94.87	99.01
410	82.49	**89.58**	94.46	86.89	94.36	99.51
495	81.01	**87.08**	92.13	87.7	94.27	99.74
511	60.96	**73.66**	80.38	75.45	91.17	99.49
551	73.7	**85**	89.37	82.14	94.74	99.61
572	40.99	**60.9**	68.1	58.99	87.64	98.01
691	57.39	**78.15**	81.42	70.08	95.43	99.43
846	71.32	**84.64**	90.89	77.94	92.5	99.34
928	64.6	**82.42**	86.67	72.72	92.77	97.55
all cams	69.56	80.56	85.35	80.01	93.15	98.77
urban	**74.54**	**83.88**	**88.57**	83.22	**93.74**	**99.01**
highway	50.99	70.95	75.87	66.06	92.17	98.76
other	69.99	76.75	81.83	83.31	91.36	97.42

model is more adapted to typical urban footage and performs better. On the bottom part of the table, the mAP at medium difficulty is slightly better for the urban scene frames, 83.88% vs 80.56% for all cameras. However, in terms of recall, all camera types show similar results at the medium and easy difficult levels. On the highway scenes, the recall values are worse at the hardest difficulty. This is justifiable by the smaller average size of annotations in this scene type, as seen in Table 4.1.

Classification Results

Table 5.5 shows the classification results for all 16 CityCam cameras. As in the previous table, below the individual camera results we present the average over all 16 cameras and the averages per scene category: urban, highway and other.

We can see in the table that the AP of the top three classes is, for the most part, slightly higher than the total one at the medium difficulty level, this indicates that the classification of these types of vehicles is better than that of the other minority classes. This corresponds to our expectations, it is likely that a deep learning detector will perform much better on the more frequent classes, especially with the degree of class imbalance this dataset faces. This is evident in Table 5.6 further below.

In the bottom part of Table 5.5 we see the results divided by traffic scene type. In bold font we highlighted the the best results for each metric, and as seen previously in the detection results, they all correspond to the urban cameras, the most prevalent in our training. The mAP for all cameras, at the

Table 5.5: Classification results on all CityCam cameras. The table shows the mean Average Precisions (mAP) at the three difficulty levels, the mean Average Precision (AP) of the three main classes at the medium level and the mean Recall (mRec) at hard and medium difficulty. The colours in the first column indicate the traffic scene type and the final rows show the average per scene type.

	mAP (%)				mRec (%)	
	s+m+l	m+l	l	top 3 cls	s+m+l	m+l
164	47.24	54.68	65.78	61.23	63.36	72.94
166	53.19	58.57	68.49	60.77	67.26	74.35
170	68.49	77.04	82.28	80.03	77.7	88.22
173	69.87	77.96	85.26	79.68	77.48	86.77
181	74.62	84.25	88.07	84.4	82.3	93.03
253	45.35	66.85	74.37	68.61	57.91	85.44
398	73.77	81.18	88.36	82	81.02	89.39
403	78.65	82.43	88.83	84.26	85.23	89.54
410	77.09	84.4	91.51	84.86	81.86	89.92
495	77.54	83.99	90.72	85.48	83.62	90.64
511	57.62	70.77	80.4	70.93	68.89	85.55
551	63.74	76.26	84.57	78.2	72.77	87.5
572	23.63	35.46	47.51	37.32	40.14	61.64
691	44.61	68.17	75.47	68.89	56.97	87.02
846	65.67	77.99	86.6	78.24	72.03	86.75
928	60.13	79.81	86.22	81.54	66.8	89.46
avg	61.33	72.49	80.28	74.15	70.96	84.89
urban	**69.74**	**79.64**	**86.62**	**80.87**	**77.25**	**88.8**
highway	37.86	56.83	65.78	58.27	51.67	78.03
other	50.22	56.63	67.14	61	65.31	73.65

medium difficulty, is 72.49% while the mAP for the urban cameras is 79.64%. For the top three classes, the mAP values are 74.15% and 80.87% for all cameras and urban respectively.

With these results, we can also confirm that the performance of the network on highway scenes is more lacking in every metric when compared to the urban cameras. We expected this behaviour due to the small portion of highway training frames in our dataset (around 16%), and also the significant difference between this scene type and the remaining data: annotations are much smaller, they tend to appear closer together and more occluded.

Table 5.6 contains the per class Average Precision for all 16 cameras. In bold font we highlight APs above 80%. We can see that only urban cameras obtained results above 80%. We can also observe by the average values that the highest scores occur on two of the three main classes: taxis and black sedans. These form around 50% of the dataset.

Table 5.7 shows the results of detection only and classification side by side, for easier comparison. All values, both mAP and recall, correspond to the medium difficulty, that is, extremely small annotations are not considered. On average, the detection task has an 8% higher mAP than classification, from 80% to 72%. The classification of the main three classes (constituting 85% of our data) is slightly higher, being 6% lower that the detection mAP. These values indicate that most of the detected vehicles are correctly

Table 5.6: Classification results on CityCam per camera and class. All values represent the mAP (%). Values above 80% are shown in bold.

	taxi	black sedan	other car	small truck	medium truck	big truck	van	medium bus	big bus	other
164	64.51	66.19	57.14	19.23	45.2	26.99	33.47	7.92	22.57	10
166	69.45	72.24	47.32	18.66	58	43.85	50.71	46.09	29.49	26.34
170	**84.19**	**82.61**	67.97	35.29	74.12	43.1	53.38	**87.12**	**90.05**	41.46
173	**88.67**	**85.39**	66.57	38.85	**83.58**	26.58	**82.01**	65.45	**86.14**	64.45
181	**86.43**	**86.73**	75.8	-	**98.48**	-	**83.68**	**90.64**	**99.85**	39.73
253	64.72	72.11	65.01	13.96	33.5	-	26.99	37.69	-	18.85
398	**85.58**	**83.71**	79.31	70.29	**82.11**	50.96	**80.97**	**83.18**	**91.11**	51.79
403	**88.33**	**85.92**	**80.3**	76.39	**85.58**	72.75	76.76	**84.61**	**81.52**	49.74
410	**88.39**	**87.82**	79.15	**82.39**	**96.01**	65.43	**85.89**	62.44	76.49	47.48
495	**89.21**	**84.64**	**84.99**	66.08	**87.15**	38.03	72.89	59.94	78.86	66.01
511	79.4	73.3	63.96	64.72	**87.64**	3.57	**82.05**	78.16	**92.11**	42.74
551	**83.47**	**80.49**	72.71	58.4	79.65	26.63	67.66	50.02	**87.99**	46.02
572	57.66	61.66	23.66	13.24	6.6	4.76	14.89	8.04	-	-
691	76.4	68.31	68.58	15.71	33.33	-	58.46	72.92	-	58.52
846	75.85	**88.39**	61.54	37.4	**85.02**	58.81	**92.05**	65.42	**97.02**	33.9
928	75.61	**83.22**	**82.87**	68.41	60.95	65.42	78.47	**87.97**	**88.24**	12.67
avg	78.62	78.92	67.3	45.27	68.56	40.53	65.02	61.73	78.57	40.65

Table 5.7: Detection and classification results side by side, per camera. All values correspond to the medium difficulty level.

	mAP (%)			mRec (%)	
	detection	classification	classification of top 3 classes	detection	classification
164	76.55	54.68	61.23	91.52	72.94
166	76.94	58.57	60.77	91.2	74.35
170	79.5	77.04	80.03	93.98	88.22
173	83.03	77.96	79.68	92.71	86.77
181	85.22	84.25	84.4	95.25	93.03
253	73.81	66.85	68.61	93.44	85.44
398	85.49	81.18	82	94.54	89.39
403	87.03	82.43	84.26	94.87	89.54
410	89.58	84.4	84.86	94.36	89.92
495	87.08	83.99	85.48	94.27	90.64
511	73.66	70.77	70.93	91.17	85.55
551	85	76.26	78.2	94.74	87.5
572	60.9	35.46	37.32	87.64	61.64
691	78.15	68.17	68.89	95.43	87.02
846	84.64	77.99	78.24	92.5	86.75
928	82.42	79.81	81.54	92.77	89.46
avg	80.56	72.49	74.15	93.15	84.89

classified by our model, especially those in the more common classes. The recall values show the same as the mAP, for detection only, the value is 8% higher than for both localization and classification, between 93% and 85%.

Figure 5.1 contains some examples of detections made on CityCam by our model. The annotated

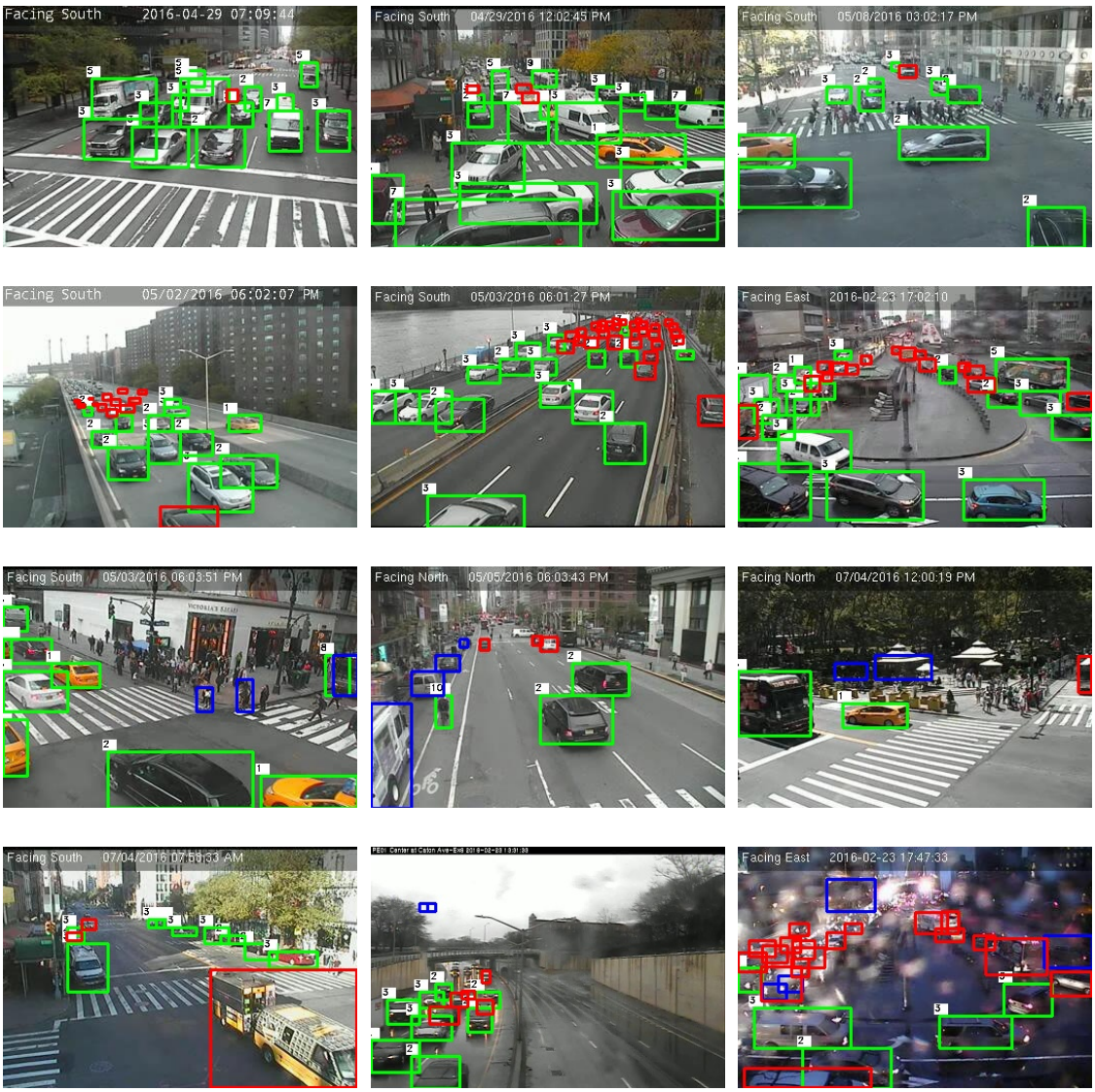


Figure 5.1: Detections made by our model on the CityCam dataset.

bounding boxes are colour coded as: green - true positive, red - false negative and blue - false positive. The top row shows three typical urban roads and intersections, most vehicles are detected, however some of the smaller and more occluded ones are missed. The second row has two highway frames and an unusual road configuration frame. As we have seen so far in the results, the performance on these scenes is less accurate. These images reveal just that. While many correct detections are made, several of the smaller vehicles are unidentified by the model. The third row contains examples of the false positives produced: pedestrians, unconventional patches on the background and some vehicles not annotated in the dataset. The final row shows an unusual vehicle that is missed and two frames affected by rain.

Effect of weather conditions

In the previous chapter, Section 4.1, we divided the CityCam data into four weather categories: normal, rain, heavy rain and shadows. We assumed that a detector would perform poorly on the images classified as *heavy rain*, however, we made no assumptions on the effects of the other types of weather conditions. To study if they in fact affect the performance of our model, we divided the test set into 5 subsets: (1) rain, (2) heavy rain, (3) shadows, (4) whole set minus rain, (5) whole set minus rain and shadows.

Table 5.8: Results of weather conditions on detection and classification. For each of the 6 data subsets, we present the mAP at all difficulty levels, the AP at medium difficulty of the three main classes and mean Recall (mRec) at the hardest and medium difficulties. The values in bold represent the best of each metric.

	mAP (%)			AP (m+l) (%)				mRec (%)	
	s+m+l	m+l	l	taxi	black sedan	other car	avg top 3 cls	s+m+l	m+l
rain	59.96	71.81	79.41	82.84	79.21	66.35	73.94	71.05	85.38
heavy rain	41.14	51.97	62.34	76.29	67.36	41.76	54.63	58.88	75.41
shadows	59.28	76.16	81.83	**86.94**	79.5	71.49	77.38	69.99	90.54
no rain	63.91	81.33	85.69	86.48	84.6	78.98	82.64	72.36	92.37
no rain and no shadows	**64.41**	**81.86**	**86.07**	86.45	**85.19**	**79.84**	**83.24**	**72.61**	**92.57**
all	61.82	77.7	83.5	85.84	83.73	76.88	81.23	70.79	89.41

Table 5.8 contains the obtained classification results. These correspond accurately to what would be expected. The worst results occur on the *heavy rain* set. As we can see, this type of rain severely affects the detection, the medium difficulty mAP is 25% lower than the mAP of the whole set. In the *rain* set, the effect is less intense. The mAP is 6% lower. The presence of shadows seems to have a minor effect, with a mAP difference of less than 2%.

Considering these three conditions had a negative influence on the performance, the best results occur on the set with no rain or shadows. Overall, the results on this set are only slightly better than those achieved on the entire testing set. This indicates that the unfavourable weather or illumination conditions don't occur frequently enough to have a high impact on the general performance scores.

Highway Results

Due to the discrepancy in results between urban cameras and highway cameras, we decided to train a model uniquely with the three highway cameras (253, 572 and 691). Table 5.9 shows the results obtained by this highway model and compares them to the ones obtained by the general one.

From these results, we see that the general model behaves better when taking into account the smaller annotations. However, the highway model provides better results at the medium a easy difficulties. The results are very similar in both implementations, but, we believe that with a larger and more extensive highway training set, the highway only model would provide more satisfying results.

Table 5.9: Comparison of an implementation trained on highway only versus the general training. The mAP is shown on th three levels of difficulty and at the medium level for the two top classes only. The mean Recall (mRec) is presented at hard and medium difficulties. In bold we show the best value between the highway model and the general one.

		mAP (%)				mRec (%)	
		s+m+l	m+l	l	top 2 cls	s+m+l	m+l
highway	253	43.51	**68.02**	73.35	**69.67**	56.73	**88.78**
	572	21.41	**37.85**	**51.84**	**39.95**	34.33	**61.94**
	691	44.01	**71.2**	**75.98**	**71.43**	56.06	**90.64**
general	253	**45.35**	66.85	**74.37**	68.86	**57.91**	85.44
	572	**23.63**	35.46	47.51	37.05	**40.14**	61.64
	691	**44.61**	68.17	75.47	68.45	**56.97**	87.02

Comparison with other models

One of the challenges of vehicle detection is the lack of benchmarks and systems to compare and evaluate models. Vehicle detection systems are very dependent on the application and most publications use private datasets.

To sidestep this issue, we decided to compare our model to state-of-the-art object detectors: Mask RCNN [30], Faster RCNN [1] and RetinaNet [60]. We are using the pre-trained implementations available on Detectron2 [61], a detector software from Facebook AI Research. All of these models have ResNet50 [23] as the backbone network and are trained on the MS COCO dataset, with over 200 thousand images.

To do this comparison we selected a random sample of 500 images from the CityCam test set and evaluated the three Detectron2 models and our implementation on the detection only task.

Table 5.10: Comparison between our implementation and other object detectors. The highest values of each metric are in bold.

		mAP			Recall		
		s+m+l	m+l	l	s+m+l	m+l	l
Detectron2	Mask RCNN	45.52	62.44	75.65	56.13	76.99	93.28
	Faster RCNN	44.69	61.23	74.47	55.61	76.19	92.66
	RetinaNet	38.88	52.98	62.88	58.72	80.02	94.87
	our	**48.54**	**70.09**	**78.36**	**58.78**	**84.88**	**94.89**

Table 5.10 contains the results obtained. Surprisingly, our model fared better on all metrics. While the Detectron2 models have not been trained on this data, they were submited to a much more extensive training and all three are the newest and best implementations of each network. The biggest difference in results occured in the mAP at the medium difficulty, where medium and large vehicles are considered. As mentioned before, this is the metric we consider the most relevant. Our implementation achieves a mAP close to 10% higher than the second best model. All other results are fairly similar. As expected, no network performs well when the smallest vehicles are considered and all achieve acceptable results if only the largest vehicles are contemplated.

5.2 Testing on Tallinn Traffic Data

We tested our fully-trained vehicle detection model on different traffic data to better understand how it would behave on other footage. For this, we are using the Tallinn Traffic dataset mentioned in 3.2. This dataset was not created for object detection and has no bounding box annotations. However, each frame is accompanied by an estimated density score (number of vehicles in image) and density map. The lack of detection annotations implies that the assessment of our model's performance was done by manual observation and by comparison to the density scores.

The main advantage of this footage lies in its temporal properties. All of the data was collected on the same month, March of 2020. On most days, the same number of photos was taken and during the same time interval. Additionally, there is footage from urban roads and intersections, highways and parking lots. It interests us to test our model on all three types of scene, focusing mainly on the typical urban road.

5.2.1 Urban Camera

We selected one typical urban scene camera to perform most of the testing on. From this particular camera we have data collected from the 9th to the 27th of March. Most of the days have data from 7 a.m. to 22 p.m., however, there are some lapses in a few days.

Figure 5.2 shows the total vehicle counts per day for this one camera. To compare the number of vehicles observed on different days, We are only including the days for which we have data on every hour. First, it is interesting to note that while this data was being collected, the Estonian government declared a state of emergency due to COVID-19, starting on the 13th of March. This justifies the much higher count of vehicles on the first weekdays of the graph (10,11 and 16), and also between the two weekends shown (14,15 and 21,22). All other weekdays (between the 18th and 27th) present a similar vehicle count.

Observing Figure 5.3 which shows the vehicle counts per two hour intervals, we can see that the trend is to have the highest number of vehicles at around 15 p.m. and the lowest after 20 p.m.

By testing our model with this data, we have been able to make the following inferences on this particular road section:

- Vehicles are much more frequent during weekdays.
- On weekdays, the number of vehicles stays relatively constant throughout the week.
- The number of vehicles is also constant on both weekend days.

This analysis is not sufficient to assert how many vehicles go through that particular road per day or per hour. This dataset only has around 6 photos per hour. However, this initial study leads us to believe

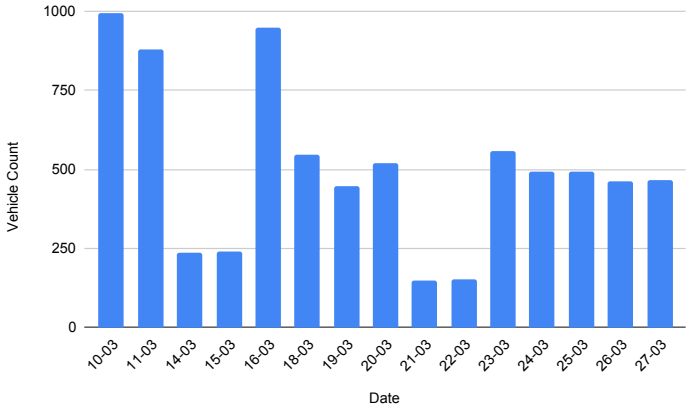


Figure 5.2: Vehicle counts per day on an urban Tallinn camera.

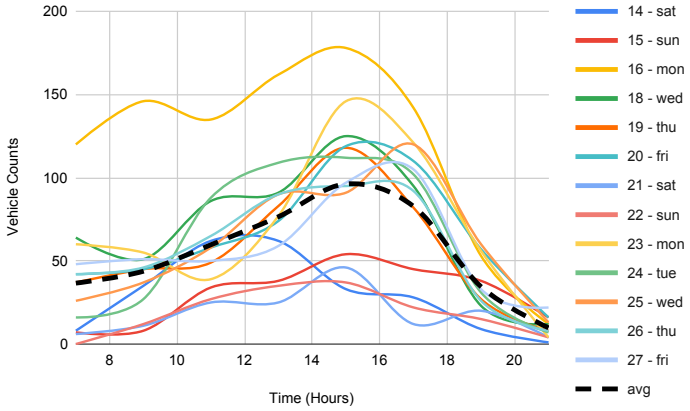


Figure 5.3: Vehicle counts per two hour intervals on an urban Tallinn camera.

that having that extra data, we could produce these results with enough accuracy.

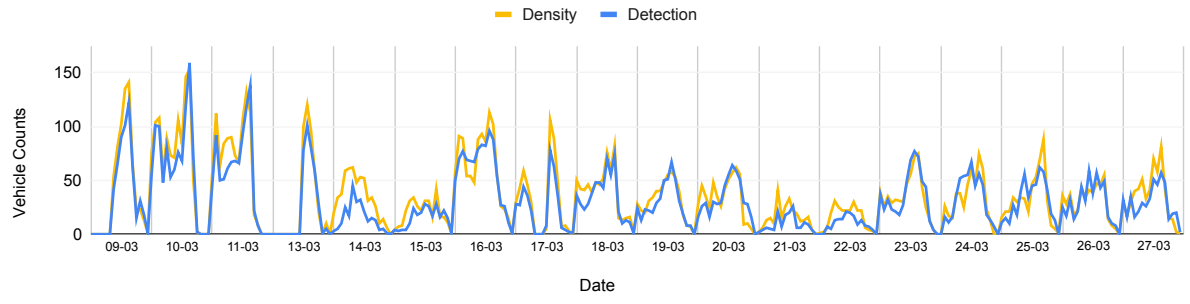


Figure 5.4: Our detection counts versus the density counts of the Tallin dataset. Both density and detection counts are plotted per hour. Each section in the horizontal axis corresponds to one day.

In figure 5.4, we compare the detection counts we obtained with our model to the existing density counts. In the graph we can see that both techniques provided similar results. This indicates that our

detection model is as effective as the density method in the task of vehicle counting. Furthermore, detection can contribute with additional information such as vehicle type, size and location.

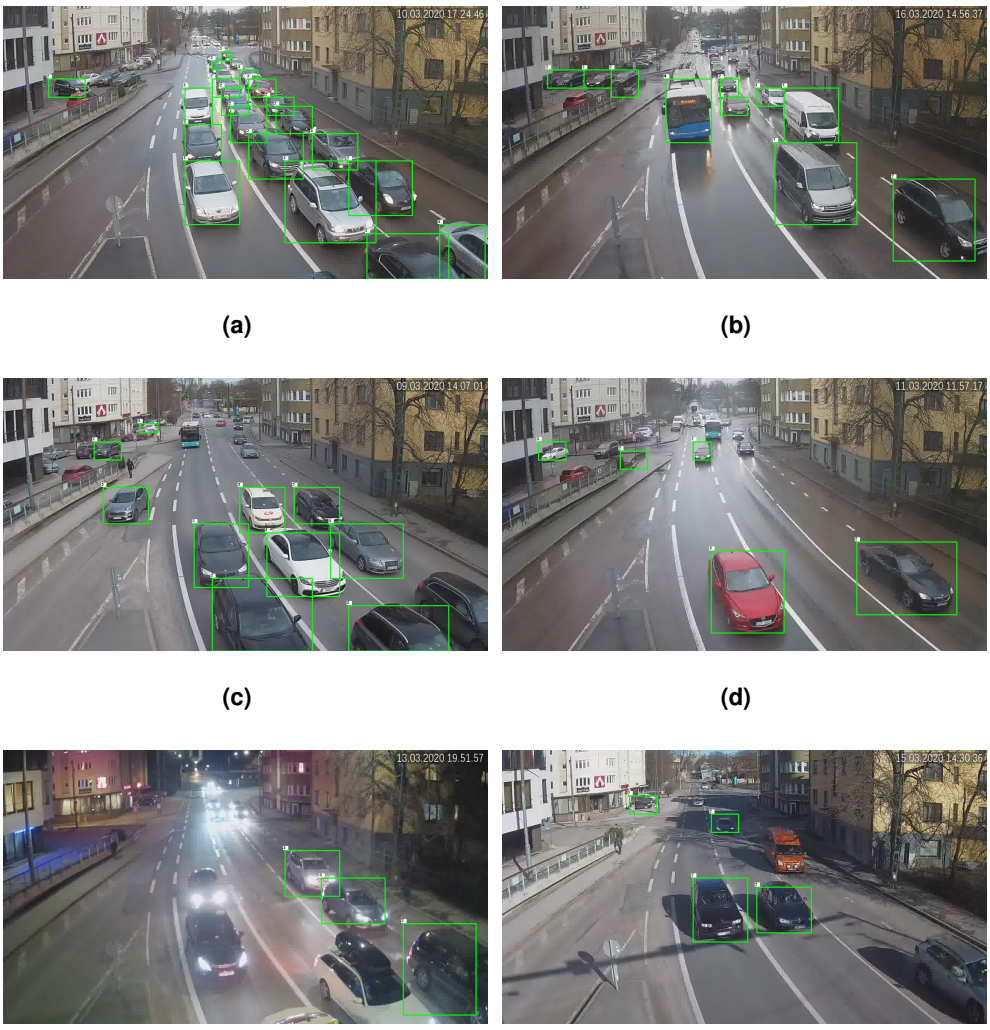
In Figure 5.5 we present some example frames obtained by applying our model to the chosen Tallinn urban traffic camera. Images 5.5(a) and 5.5(b) show frames where the model provided the desired results. Most vehicles are detected, especially in the area closest to the camera. Also, in 5.5(b) both the bus and the vans are correctly classified, as are the more typical cars. In 5.5(c) we can see a reoccurring false negative, the car in the lower right corner of the image was not detected. This happens often to vehicles that are truncated by the frame's margins. The next figure, 5.5(d), shows a false positive on the upper left that was detected more than once on this camera's footage. This type of false positives can be eliminated by hard negative mining which would require training data. Another option is to define an image section representing the road and solely detect on that region. Figures 5.5(e) and 5.5(f) show the behaviour of the model on different illumination conditions: night and shadows, respectively. This model has not been trained with nighttime footage, as such it is not expected to properly detect vehicles with this type of illumination. As we can see, the performance is quite inferior. With the presence of shadows, the performance seems to decrease, however not as significantly.

5.2.2 Highway Camera

To test our model on highway scenes, we selected one highway camera from the Tallinn dataset. Our goal with this test is not to perform temporal inferences based on the number of vehicles per day or per hour as before. We simply intend to assess our model on a different road type. For this reason we are only computing the detections for one day.

As we saw previously when testing on CityCam, our model has a slightly worse performance on highway scenes. The same occurs on the Tallinn data tested. Figure 5.6 shows the comparison between our detection counts and the data's density vehicle counts. Our model tends to detect less vehicles than the density method.

Figure 5.7 contains four frames from this highway camera with our detected bounding boxes in green. As expected, our model misses the smallest vehicles, the ones furthest from the camera. It also frequently missed vehicles truncated by the margins of the image, as in the right margin of Figure 5.7(b). With this being a highway camera, there was a higher number of trucks in the frames. These were difficult to detect by our model. This type of vehicle is infrequent in our training data and their dimensions are unusually large. In Figures 5.7(c) and 5.7(d) we can see how these vehicles were not detected, while the regular cars around them were.



(a) **(b)**

(c) **(d)**

(e) **(f)**

Figure 5.5: Detections made by our model on the Tallinn traffic data.

5.2.3 Parking Lot camera

The Tallinn dataset contains parking lot cameras. This is a type of scene not previously seen by our model during training nor testing. We chose to analyse two different days from the same parking lot camera. Both days were mondays, one before the COVID-19 state of emergency and one after.

This camera captures an unusual scene for our model, as such, we expect the results to be slightly less reliable. Figure 5.8 compares the density vehicle counts and our detection counts. As we can see, there is a discrepancy in the number of vehicles detected by each method. However, it is still evident by our counts that several more vehicles occupied the parking lot on the first day (before quarantine) versus the second one (after quarantine).

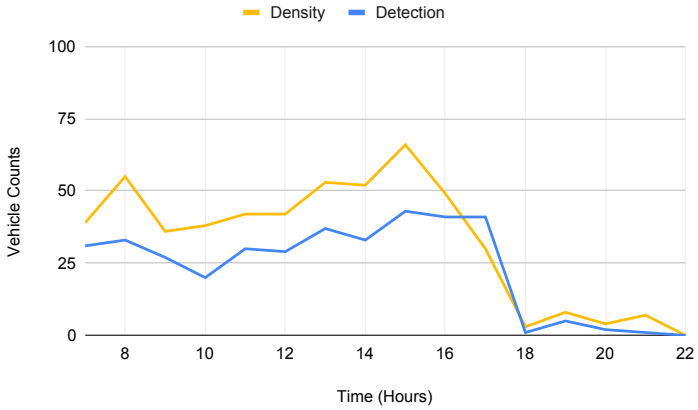


Figure 5.6: Detection vs density vehicle counts on a highway camera from the Tallinn traffic dataset.

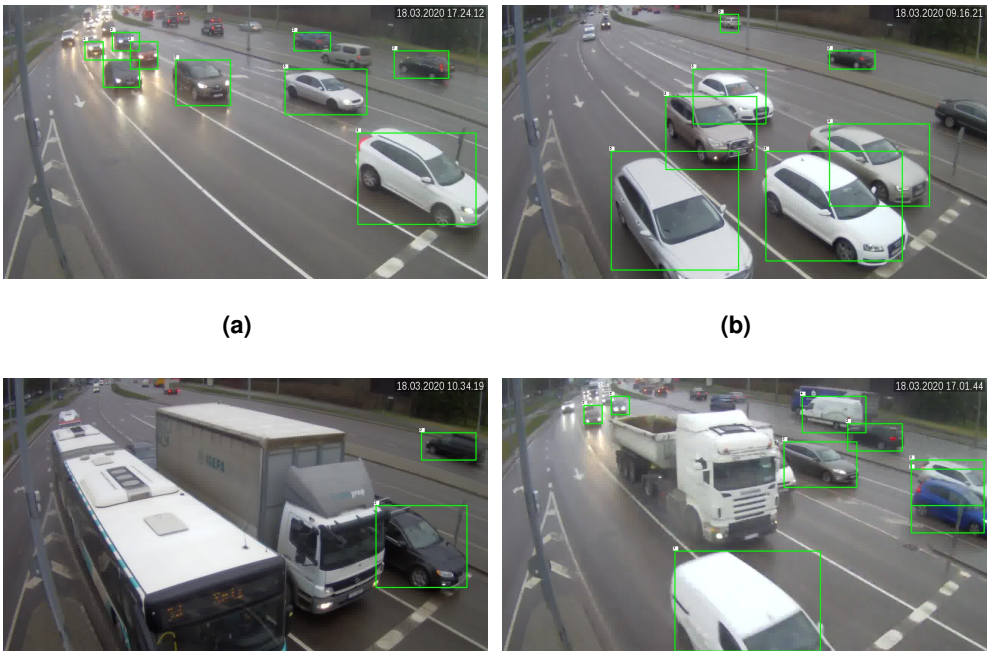


Figure 5.7: Detections made by our model on the Tallinn highway traffic data.

Figure 5.9 shows four frames from the parking lot camera with our detections. The first frame, 5.9(a), shows a very cluttered and full view. Our model behaved poorly on such a scene, missing several vehicles. This is not surprising considering the amount of occlusion the cars present. On the next frame, 5.9(b), there are considerably less vehicles and the detections appear to be more accurate. The same can be said for frame 5.9(c) where all parked cars are correctly detected. On the last frame, 5.9(d), we can see the performance of our model at night time. It is surprising that some of the vehicles are

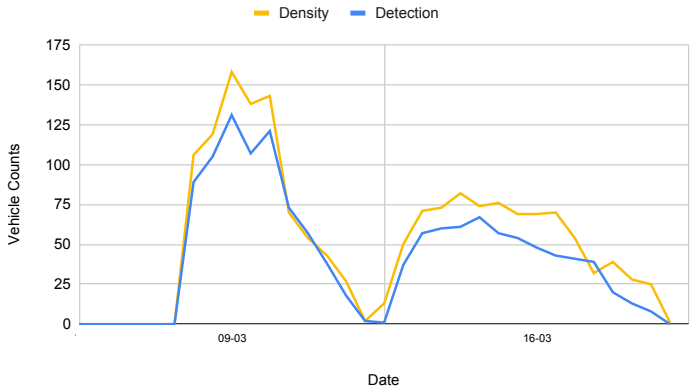


Figure 5.8: Detection vs density vehicle counts on a parking lot camera from the Tallinn traffic dataset.

detected given that none of the training data presents nighttime illumination.

(a) **(b)**

(c) **(d)**

Figure 5.9: Detections made by our model on the Tallinn parking lot traffic data.

6

Conclusions

Contents

In this chapter we present and discuss our main conclusions on the work developed. Lastly, we introduce possible lines of future work in this area.

6.1 Discussion

The aim of this project was to assess the use of object detection to perform the task of vehicle counting on typical low resolution traffic data. To accomplish this, we adapted and trained a Faster RCNN model on the CityCam dataset and later tested it on this same dataset and on traffic footage from the city of Tallinn.

We found that one of the main aspects that affects the results of our model is the road type. Most of the training data used represents urban streets and intersections. As such, the performance is much better in this type of setting versus a highway road, as we saw in both CityCam and the Tallinn data. Similarly, we also found that the results are better at daytime and with mild weather conditions. We are certain that all of these issues would be solved if more training data for these situations were available.

Besides this, we also found that the size of the vehicles has a great influence on the outcome. Smaller vehicles, as expected, are less frequently detected. However, if we are using this method to count vehicles in a video sequence, it is not necessary to find the smallest vehicles in each frame, as they will be larger in the ones that follow.

Finally, we found the counting results on the Tallinn urban footage very comparable to the ones obtained through density estimation. We thus conclude that both techniques are equally effective for vehicle counting and detection can provide additional information such as size, location and class.

With this, we conclude that vehicle counting can be performed by a state-of-the-art object detector on low resolution traffic data. However, this method is not advisable if there is a necessity in detecting the smallest of vehicles. Due to the construction of object detectors, extremely small objects are missed or ignored. Additionally, this method requires some domain adaptation. Urban cameras and highway cameras produce considerably different scenes, as such, to perform well on both, the model has to be trained on both. The same can be said for unusual roads such as roundabouts and different illumination and weather conditions, within reason.

6.2 Future Work

In this work we attempted to perform vehicle counting through object detection which is usually done with density estimation. Both these techniques have flaws, mainly when counting small vehicles. And both have different advantages. Density techniques are more accurate on counting occluded vehicles while detection provides the location of each vehicle. An interesting approach to the vehicle counting task

would be to combine the detector we implemented to prior knowledge given by density estimation. We expect this would improve the detectors performance, especially when dealing with occluded vehicles.

Bibliography

[1] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[3] M. Loukadakis, J. Cano, and M. O'Boyle, "Accelerating deep neural networks on low power heterogeneous architectures," 2018.

[4] R. Guerrero-Gómez-Olmedo, B. Torre-Jiménez, R. López-Sastre, S. Maldonado-Bascón, and D. Onoro-Rubio, "Extremely overlapping vehicle counting," in *Iberian Conference on Pattern Recognition and Image Analysis*. Springer, 2015, pp. 423–431.

[5] S. Zhang, G. Wu, J. P. Costeira, and J. M. Moura, "Understanding traffic density from large-scale web camera data," *arXiv preprint arXiv:1703.05868*, 2017.

[6] E. Commission, *Intelligent transport systems - EU-funded research for efficient, clean and safe road transport*. Luxembourg: Publications Office of the European Union, 2010.

[7] J. Guerrero-Ibáñez, S. Zeadally, and J. Contreras-Castillo, "Sensor technologies for intelligent transportation systems," *Sensors*, vol. 18, no. 4, p. 1212, 2018.

[8] M. Bommes, A. Fazekas, T. Volkenhoff, and M. Oeser, "Video based intelligent transportation systems–state of the art and future development," *Transportation Research Procedia*, vol. 14, pp. 4495–4504, 2016.

[9] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[10] Z. Zou, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *arXiv preprint arXiv:1905.05055*, 2019.

[11] L. Liu, W. Ouyang, X. Wang, P. W. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *CoRR*, vol. abs/1809.02165, 2018. [Online]. Available: http://arxiv.org/abs/1809.02165

[12] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of physiology*, vol. 160, no. 1, pp. 106–154, 1962.

[13] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2. Ieee, 1999, pp. 1150–1157.

[14] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, vol. 1. IEEE, 2001, pp. I–I.

[15] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[16] P. Felzenszwalb, D. McAllester, and D. Ramanan, "A discriminatively trained, multiscale, deformable part model," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2008, pp. 1–8.

[17] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, pp. 1627–1645, 2009.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[19] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[20] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[21] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[22] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[24] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[25] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[26] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *International journal of computer vision*, vol. 104, no. 2, pp. 154–171, 2013.

[27] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (VOC) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *CoRR*, vol. abs/1406.4729, 2014. [Online]. Available: http://arxiv.org/abs/1406.4729

[29] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: object detection via region-based fully convolutional networks," *CoRR*, vol. abs/1605.06409, 2016. [Online]. Available: http://arxiv.org/abs/1605.06409

[30] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017. [Online]. Available: http://arxiv.org/abs/1703.06870

[31] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[32] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[33] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *arXiv preprint*, 2017.

[34] ——, "YOLOv3: An incremental improvement," *arXiv*, 2018.

[35] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[36] Z. Yang and L. S. Pun-Cheng, “Vehicle detection in intelligent transportation systems and its applications under varying environments: A review,” *Image and Vision Computing*, vol. 69, pp. 143–154, 2018.

[37] Z. Sun, G. Bebis, and R. Miller, “On-road vehicle detection: A review,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 5, pp. 694–711, 2006.

[38] N. Buch, S. A. Velastin, and J. Orwell, “A review of computer vision techniques for the analysis of urban traffic,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 3, pp. 920–939, 2011.

[39] G. Yan, M. Yu, Y. Yu, and L. Fan, “Real-time vehicle detection using histograms of oriented gradients and adaboost classification,” *Optik*, vol. 127, no. 19, pp. 7941–7951, 2016.

[40] X. Wen, L. Shao, W. Fang, and Y. Xue, “Efficient feature selection and classification for vehicle detection,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 3, pp. 508–517, 2014.

[41] H. Fu, H. Ma, Y. Liu, and D. Lu, “A vehicle classification system based on hierarchical multi-svms in crowded traffic scenes,” *Neurocomputing*, vol. 211, pp. 182–190, 2016.

[42] C. Stauffer and W. E. L. Grimson, “Adaptive background mixture models for real-time tracking,” in *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, vol. 2. IEEE, 1999, pp. 246–252.

[43] Z. Chen, T. Ellis, and S. A. Velastin, “Vehicle detection, tracking and classification in urban traffic,” in *2012 15th International IEEE Conference on Intelligent Transportation Systems*. IEEE, 2012, pp. 951–956.

[44] S. Sivaraman and M. M. Trivedi, “Active learning for on-road vehicle detection: A comparative study,” *Machine vision and applications*, vol. 25, no. 3, pp. 599–611, 2014.

[45] M. Atibi, I. Atouf, M. Boussaa, and A. Bennis, “Real-time detection of vehicle using the haar-like features and artificial neuron networks,” *Proc. Computer Science*, vol. 73, pp. 24–31, 2015.

[46] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the KITTI vision benchmark suite,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[47] Q. Fan, L. Brown, and J. Smith, “A closer look at faster R-CNN for vehicle detection,” in *2016 IEEE intelligent vehicles symposium (IV)*. IEEE, 2016, pp. 124–129.

[48] A. Tourani, S. Soroori, A. Shahbahrami, S. Khazaee, and A. Akoushideh, "A robust vehicle detection approach based on faster R-CNN algorithm," in *2019 4th International Conference on Pattern Recognition and Image Analysis (IPRIA)*. IEEE, 2019, pp. 119–123.

[49] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3d object representations for fine-grained categorization," in *Proceedings of the IEEE international conference on computer vision workshops*, 2013, pp. 554–561.

[50] L. Huang, W. Xu, S. Liu, V. Pandey, and N. R. Juri, "Enabling versatile analysis of large scale traffic video data with deep learning and HiveQL," in *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2017, pp. 1153–1162.

[51] H. Song, H. Liang, H. Li, Z. Dai, and X. Yun, "Vision-based vehicle detection and counting system using deep learning in highway scenes," *European Transport Research Review*, vol. 11, no. 1, p. 51, 2019.

[52] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to sift or surf," in *2011 International conference on computer vision*. Ieee, 2011, pp. 2564–2571.

[53] V. Lempitsky and A. Zisserman, "Learning to count objects in images," in *Advances in neural information processing systems*, 2010, pp. 1324–1332.

[54] L. Fiaschi, U. Köthe, R. Nair, and F. A. Hamprecht, "Learning to count with regression forest and structured labels," in *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*. IEEE, 2012, pp. 2685–2688.

[55] Z. Dong, Y. Wu, M. Pei, and Y. Jia, "Vehicle type classification using a semisupervised convolutional neural network," *IEEE transactions on intelligent transportation systems*, vol. 16, no. 4, pp. 2247–2256, 2015.

[56] "Keras-fasterrcnn." [Online]. Available: https://github.com/you359/Keras-FasterRCNN

[57] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks." [Online]. Available: https://github.com/ShaoqingRen/faster_rcnn

[58] Y. Jia, "Caffe: An open source convolutional architecture for fast feature embedding." [Online]. Available: http://caffe.berkeleyvision.org/

[59] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[60] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, “Focal loss for dense object detection,” *CoRR*, vol. abs/1708.02002, 2017. [Online]. Available: http://arxiv.org/abs/1708.02002

[61] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, “Detectron2,” https://github.com/facebookresearch/detectron2, 2019.

A

Concepts

A.1 Concepts

In this section, some of the basic and reoccurring concepts of object detectors are presented. The contents are organized in alphabetical order for reading simplicity.

Bounding Box - rectangular frame that surrounds an object. In detection problems it is usually 2D, however, it can be a 3D box in some applications. It is represented by a set of coordinates. A bounding box is the most typical representation of the location of an object, figure 2.2 shows an image labeled with bounding boxes.

Bounding Box Regression - method used to refine bounding box predictions. Usually, the initial predictions are created in a less accurate and less computationally intense way, hence, it is customary to apply a linear regression to an already classified bounding box to better locate an object.

Data Augmentation - technique to obtain a larger quantity of data by applying minor transformations to an existing dataset. Transformations such as rotations, translations, scale and illumination changes are among the most common. An object detection algorithm should be invariant to these changes, therefore,

an original image and one with transformations should provide the same classification result while still being considered as distinct instances by the detector. Another useful data augmentation method is to sample patches of the original image, causing the detection to be performed at a different scale of the same object.

Hard negative mining - in object detection there tends to be a significant unbalance between positive instances (objects) and negative instances (background or non-objects) in data, implicating it might not be enough to train a classifier with only positive examples. Thus, random bounding boxes around background patches are created as an example of negative instances so that the classifier can learn both positives and negatives. When testing, if the classifier returns false positives, these are added to the set of negatives, substantially reducing the number of false positives, *i.e.* background regions classified as objects.

Hyperparameter - type of parameter used in learning algorithms that is set prior to the execution of the learning process. This term is used to distinguish between these fixed parameters and the ones learned by the algorithm.

Image Gradient - a gradient is a vector that points in the direction of greatest increase of a function. In computer vision, it is applied to an image to discover the regions with the most significant change in colors (it is common to apply it to grayscale images).

Intersection over Union (IoU) - measure of overlap between two bounding boxes. It is used as an evaluation metric to measure the accuracy of object detectors. Having two bounding boxes, their area of overlap (intersection) and area of union are computed, they are then divided giving

$$IoU = \frac{Area\,of\,Intersection}{Area\,of\,Union} \tag{A.1}$$

For evaluation purposes, a ground-truth bounding box, which is a hand-labeled one, is compared to the predicted bounding box. To determine whether a prediction is suitable, a threshold is applied to the IoU score, usually, an IoU greater than 0.5 is considered acceptable.

Non-Maximum Suppression - technique employed to fix duplicate object predictions produced by detectors. After the bounding box proposals and classifications are made, duplicates are found by calculating the IoU between every two objects of the same class. If the IoU overlap is greater than a threshold, usually set at 0.5, the prediction with the lowest confidence score is eliminated. Effectively dealing with multiple detections of the same object.

Sliding Window and Image Pyramid - these are two techniques used in object detection to localize objects in images at multiple scales and locations. A sliding window is a rectangular region of a certain size that is slid across an image to obtain all regions of that same size. The classification is done on

each region separately, thus allowing the detector to know the location of the object found. The image pyramid is, as the name implies, several images stacked together, each smaller than the last, every pyramid level is a re-sampling of the original image (usually to a smaller size). By applying the sliding window technique to an image pyramid, it is possible to obtain objects' locations at different scales.