DELFT
UNIVERSITY
OF TECHNOLOGY
Department of Computer Science

High-Speed Object Detection

Design, Study *and* Implementation

Master's Thesis

Tom Runia

August 2015

Delft University of Technology

Master's Thesis

High-Speed Object Detection

Design, Study and Implementation

of a

Detection Framework using Channel Features and Boosting

Author
Tom Runia

Thesis Committee
Prof. dr. ir. M.J.T. Reinders
Prof. dr. M. Loog
Prof. dr. A. Hanjalic

Supervisors
Ir. R. Lukassen
Dr. ir. L. Zhang

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science

in the

Computer Vision Lab
Department of Pattern Recognition and Bioinformatics

24th August, 2015

Abstract

In this thesis we design, implement and study a high-speed object detection framework. Our baseline detector uses integral channel features as object representation and AdaBoost as supervised learning algorithm. We suggest the implementation of two approximation techniques for speeding up the baseline detector and show their effectiveness by performing experiments on both detection quality and speed. The first improvement to our baseline classifier focuses on speeding up the classification of subwindows by formulating the problem as sequential decision process. The second improvement provides better multiscale handling to detect objects of all sizes without rescaling the input image. This speed-up builds upon the scale invariance property of image statistics in natural images that offers a powerful relationship for approximating feature responses of adjacent scales. While these techniques are not new itself, to our best knowledge we are the first to combine these into a framework for high-speed object detection. Our detection framework is built from the ground up using a fast GPU implementation. Based on these approximation techniques and the GPU implementation for extracting channel features we report detection speeds of 55 fps on a laptop. In a series of experiments we study the contribution of each component to the overall detection time and the possible change in detection quality due to the approximations. We train and test the detector on our car dataset that was constructed for this work. More specifically we focus on rear-view car detection. However the methods discussed are not limited to this object class.

Thesis Committee

Chair	Prof. dr. ir. M.J.T. Reinders	EEMCS faculty, TU Delft
University Supervisor	Prof. dr. M. Loog	EEMCS faculty, TU Delft
Company Supervisor	Ir. R. Lukassen	Product Lab, TomTom N.V.
University Co-Supervisor	Dr. ir. L. Zhang	EEMCS faculty, TU Delft
Committee Member	Prof. dr. A. Hanjalic	EEMCS faculty, TU Delft

Preface

After having obtained a bachelor degree in Applied Physics I came to realization that my passion was not in physics. My main interests are within the field of Computer Science, more specifically artificial intelligence, machine learning, computer vision and image processing. Therefore I decided to pursue a master degree in Computer Science with the courses chosen around those interests. Indeed my favorite courses turned out the be the ones on pattern recognition, machine learning and computer vision. Deciding upon a research group for my graduation project was a straightforward choice: the Pattern Recognition and Bioinformatics group at the faculty for Electrical Engineering, Mathematics and Computer Science.

There was the option to combine my graduation project with an internship at a company. Given the fact that I had never taken an internship before this possibility directly appealed to me. I also believe that applying knowledge to real-world applications is important when pursuing an engineering degree from a university of technology. Following up on the decision to do my graduation project at a company I spoke to three different companies to explore the opportunities. For me TomTom was the company that offered me the best possibilities. In agreement with Roland van Venrooij at TomTom we decided that I would do my thesis project and internship at the research lab of TomTom in Eindhoven. During the nine month period in which this thesis was written, I was an intern researcher in the Product Lab. This group focuses on inspiring the company with new product opportunities for the 2-4 year road map. For me this was the perfect environment to pursue my interest of researching computer vision and machine learning in an environment with direct practical application.

Within the Product Lab I found Robert Lukassen enthusiastic to be my day to day supervisor. Robert is a principal architect at TomTom focusing on computer vision, crowd-sourcing and positioning research. At the TU Delft both Prof. Dr. Marco Loog and Dr. ir. Lu Zhang from the PRB group accepted the request of becoming my supervisors. During the course of this project my weeks were more or less divided in half between working at the research lab in Eindhoven and the university in Delft.

Acknowledgments

First and foremost I offer my sincerest gratitude to my daily supervisors Robert Lukassen and Marco Loog. Robert, thank you for being my daily supervisor at TomTom. In the months I have spent at TomTom you have taught me an enormous amount of valuable information and skills. Most notably the use of computer vision in real-world applications, low-level hardware designs, software parallelization, the possibilities of OpenCV and product development are valuable additions to my skill set. Marco, thanks for being the best professor I could have wished for. Every week I was looking forward to our discussions on machine learning, computer vision and the research community in general. I am grateful for all your mathematical insights, critical questions, valuable feedback and your ability to keeping me focused on asking the right questions.

For her incredible expertise and knowledge of computer vision, I would like to thank Lu Zhang. Thank you for all the fruitful discussions we have had, the recommendations on excellent computer vision papers that have helped to shape this thesis and all your feedback throughout the project. To my committee members Marcel Reinders and Alan Hanjalic, I am extremely grateful for taking the time to read my thesis and attending my thesis presentation.

At TomTom I wish to thank all the people of the Product Lab. In particular I am grateful towards Roland van Venrooij who invited me to do my project at TomTom and made everything possible. I would also thank Roland and Martijn Mortier for organizing biweekly meetings with the crowd-sourcing team to share project updates and have interesting discussions on focus points and upcoming products. Regarding my internship at TomTom I would also like to thank Remy Alidarso and Robin van den Berg for bringing me in contact with Roland before our first meeting.

I would also like to thank Liesbeth and Jan for having me as guest in their house, without you this project would have been more stressful and less fun! Thanks for a key to your house, the excellent dinners every night and for borrowing the bicycle. Finally, the biggest gratitude is for my parents who gave me the incredible opportunity to obtain an engineering degree! All these people have played a crucial role helping me improve as an engineer, a scientist and person over the past year.

Tom Runia
Delft, The Netherlands
24th August, 2015

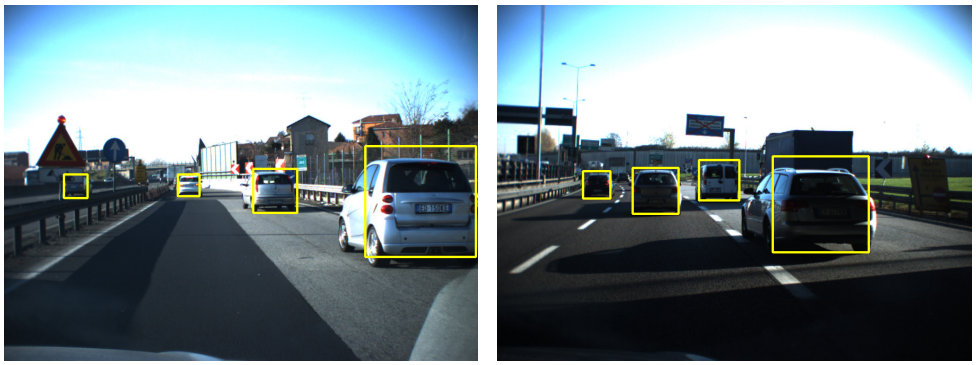
Table of Contents

1 Introduction

Computers have become the most important technology in our recent lives and are capable of solving increasingly difficult problems. More specifically, computers are generally better than humans in repetitive, data intensive and computational tasks. Over the last decade computers have become so powerful that they can be used to perform complex tasks such as natural language processing, high-level visual understanding of photographic scenes and navigation for self-driving cars. This thesis is about one of these highly computational applications that has become possible in recent years. The problem of interest for this research is *object detection*: the complex task of finding objects in a given image or video frame. In this first chapter we motivate the importance of this problem, formulate our research goal, define the scope of our work and summarize our contributions.

1.1 Background

As briefly mentioned in the preface of this work, the current research is conducted at TomTom Product Lab in Eindhoven. TomTom and the automotive and navigation industries in general, show great interest for computer vision technologies [8, 36]. More and more cars are equipped with cameras that can perceive and analyze the world around the vehicle for various purposes. High-resolution cameras and powerful hardware platforms are rapidly becoming inexpensive, opening up a wide range of new applications. In the context of Advanced Driver Assistance Systems (ADAS), computer vision technologies are designed for solving problems such as forward collision detection [61], obstacle detection [87], lane guidance [20], traffic sign recognition [55] and pedestrian detection [11]. At TomTom however, the computer vision research is *not only* focused on complete safety-grade autonomous driving systems, but rather the research concentrates on crowd-sourcing applications and designing new features for consumer products. Having this in mind, there is great interest for running computer vision algorithms on low-power devices with inexpensive hardware such as embedded platforms or smartphones. Not only is running computer vision and machine learning applications on such platforms



(a) Occlusions **(b)** Difficult lighting

Figure 1.1: Car detection for highway scenarios [8].

challenging in itself, there are also scenarios in which multiple algorithms are running in parallel on a single device. This motivates us to study highly optimized computer vision algorithms, and more specifically the problem of object detection.

The ability of running object detection on smartphones and embedded platforms opens up a wide range of possibilities. We foresee applications related to crowd-sourcing map data by aggregating vision based detections, providing forward collision warnings to drivers or recognizing speed limits, traffic signs and parking places. To limit the scope of our current work we choose to study the problem of *rear-view car detection*. However the methods discussed are certainly not limited to this object class. To study this object detection problem we analyze videos captured by cameras mounted on the dashboard of a driving car of which two example images are displayed in Figure 1.1. The actual setup is not so interesting, since we simply focus on performing general object detection in video or static images. Because the final goal is to run our detector on embedded devices we are obsessed with the speed, runtime and computational efficiency of our algorithms. We aim for high-speed detections or real-time performance. We state that for the task of car detection, *real-time* is a flexible term. In our terminology, real-time operation stands for fast enough in order to not miss single objects that moves through the scene, irrespective of the objects speed. This short introduction leads us to formulating our research question.

1.2 Research Goal

The research goal of this thesis is to design, study and implement a high-speed object detection framework. We focus on improving the speed of rear-view car detection, while providing state-of-the-art detection quality. Our goal is to learn from the design process, identify and solve possible bottlenecks in detection speed and recognize important factors contributing to the overall detection quality. Based on the lessons we learn during the design process we provide a number of proposals for future research.

1.3 Scope and System Setup

This is a work on visual object detection in digital images captured by a monocular camera. We do not consider additional sensor data such as radar, lidar or GPS. We limit ourselves to object detection using rigid-templates in a sliding-window framework. Each image or video frame is analyzed "as is", meaning that no contextual information, motion vectors, tracking information, scene geometry, depth maps or additional prior knowledge are available. These tight restrictions enable practical applications beyond the scope of car detection. For example we might also want to detect traffic signs which typically move quickly through the field-of-view of the camera and can only be registered if the object detector runs at high-speed without incorporating knowledge from previous frames. We study object detection as a classical supervised learning problem; the detectors we propose are trained on a large dataset of car images. During the detection phase the trained models are used for performing binary classification of subwindows in an image.

Throughout this thesis we will report experimental results also including speed benchmarks, therefore we believe it is appropriate to give an overview of the hardware specifications that are used for achieving the results. The hardware setup is kept as simple as possible. We use a simple monocular camera, for example one that can be found in a smartphone. To analyze the images we use straightforward computing platforms consisting of a CPU and GPU for analyzing the input images. Table 1.1 presents an overview of the hardware platforms used through this research. Experiments for this thesis are performed on the MacBook Pro (late 2013). The powerful desktop machine is solely utilized for training classifiers, which generally requires powerful hardware that can run continuously for multiple hours. We refer to these hardware platforms as `Platform I` and `Platform II`.

Our detector is originally designed to be as efficient and fast as possible, having in mind that it can run on embedded devices. While we demonstrate a very fast detector, unfortunately we were unable to run our detector on an embedded device within the course of this project. Our specific embedded platform is the Jetson TK1 platform featuring the NVIDIA Tegra K1 processor. However NVIDIA has not (yet) released OpenCL support for the Tegra K1 platform (Linux for Tegra) making it impossible to run our GPU parallelized code on the embedded platform.

1.4 Overview of Contributions

Our foremost contribution is the design and implementation of a complete framework for high-speed object detection. We start from scratch by motivating our baseline detector which we then gradually evolve into a high-speed object detector. This involves discussion of image processing, machine learning, computer vision and computational speedups. The system we propose runs fully autonomous and enables any device with a camera to perform object detection, thus making it possible for integration into an existing system.

Table 1.1: Hardware specification of the platforms for this research.

	Platform I (laptop)	**Platform II** (desktop)
Operating System	Mac OS X 10.9	Ubuntu 14.04
CPU	Intel Core i5-4258U	Intel Core i7-2600
CPU frequency	2.4 GHz	3.4 Ghz
GPU	Intel Iris 3000	AMD Radeon HD6450
GPU frequency	1.2 GHz	625 Mhz
Compute Units	4 + 280	8 + 160
Memory	8 GB	16 GB
OpenCL version	1.2	1.1

The underlying motivation for designing a complete detection framework is to obtain better understanding of what components contribute to making a detector fast and what design decisions have negative effect on detection quality. Our work combines a number of ideas from existing computer vision literature to construct a highly efficient framework for object detections. Furthermore we develop efficient implementations to take full advantage of powerful GPU architecture and suggest additional improvements in the detection architecture.

Based on a review of existing literature on time constrained object detection we select methods that find the right balance between speed and detection rate. Our detector builds upon a foundation of integral channel features [16] as object descriptors and uses a boosting framework reminiscent of Viola and Jones [88] for feature selection and constructing a strong classifier. With our baseline detector as starting point we propose the implementation of two approximation techniques for speeding up the detector. Our first improvement focuses on the classification of subwindows by including soft-cascade. More specifically we choose WaldBoost [79] which formulates the classification problem as sequential decision process making decisions based on a statistical ratio test. The second improvement targets better multiscale handling. As we will see, constructing the image pyramid for multiscale analysis is a time consuming process. We propose the implementation of a feature approximation technique [13] for performing multiscale object detections without image rescaling [3]. The core contribution of our work is the integration of these high-performance vision algorithms into a full detection framework and the analysis of the contributions of these techniques to high-speed performance. In other words, we show what algorithms yield the most important contributions to achieving real-time detection speeds and potential sacrifices in detection accuracy. We also discuss the technical aspect of our detection framework by giving a number of implementation details.

The work by Beneson *et al.* [3] shares the most overlap with our work. Similar to our work the authors propose a very fast object detector, however they focus on the problem of *pedestrian* detection. In their detection framework, core contribution for reaching high detection speeds is a ground place estimation technique for reducing the search space of

images. The authors report detection speeds of 100 fps, using the ground plane estimation technique requiring a stereo-camera. Using the stereo-camera setup and the ground plane estimation technique the authors are able to reduce the search space to 640×60 pixels regions over 10 scales, hence this allows them to run the detector at such high speeds. Our work is different since we do not consider ground plane estimation but rather focus on speeding up the classification of subwindows and we study the problem of car detection. Below we give a summary of our contributions.

- **High-Speed Object Detector.**[1] In this work we suggest and implement one of the fastest object detectors available in current computer vision literature. Building on a solid foundation of integral channel features in a boosting framework we suggest two approximation techniques for speeding up the detection process without a decrease in detection quality. These algorithmic speed-ups are accompanied by a number of computational improvements including a fast GPU implementation for extracting image channels and CPU parallelization for rapidly scanning the input image for objects. For images 640×480 at 25 scales our detector runs at 55 fps on a modern laptop. Reducing the search space to $640 \times 200 \cdot 25$ scales increases the detection speed to 150 fps.
- **Performance Study.** Starting with our baseline detector we identify the bottlenecks in the detection process and propose the implementation of two approximation techniques for speeding up our detector. These techniques are not new in itself but to our best knowledge we are the first to combine these into one detection framework.
- **Insightful Experiments.** Given our detection framework, we present a large amount of experiments revealing a number of interesting results. We provide new insights in the problem of rear-view car detection and show that the classification problem can be "solved" with a limited set of very simple features focusing on horizontal edges in an image. We show the contribution and effect of the approximation techniques to both detection quality and detection speed. The experiments show that our soft-cascade and improved multiscale handling are essential in achieving high detection speeds with limited effect on the detection rate. In fact we show that detection rate increases by performing multiscale detections without image rescaling. Throughout the research we also present a number of visualizations to increase insight in how the detector operates and spends its computational resources.
- **TomTom Dataset.** For this work we have put together a dataset for rear-view car detection by data mining of TomTom street view dataset recorded by mobile mapping vehicles. Our training data consists of 2000 rear-view car samples extracted from panorama images captured by TomTom mobile mapping vehicles. Random we have sampled a large collection of panoramas captured by the fleet of mapping cars. From these panoramas all rear view cars and vans are manually labeled and bounding boxes are extracted at size 80×80 to serve as training data for our detector. Additionally 4000 random patches, not containing cars or vans, serve as negative training data. Figure 1.2 shows a subset our training examples.

[1] The code repository and rear-view car dataset are property of TomTom and will not be made public.

Figure 1.2: Our TomTom dataset for rear-view car detection.

1.5 Thesis Organization

In this brief introductory chapter we have outlined the context of this work, the motivation of studying high-speed object detection, research objective and a summary of contributions. The rest of this thesis is centered around our object detector. After introducing the baseline detector we identify the bottlenecks regarding speed and gradually propose new techniques for improving the detector. For each of the techniques we suggest we provide in-depth theoretical discussion and motivation. After discussing the individual components of the detector, we describe our overall detection framework and perform a series of experiments. More precisely the chapters in this thesis are organized as follows.

- In **Chapter 2** we give an introduction to object detection and discuss related work. Readers familiar with theory related to object detection may decide to skip the first two sections of this chapter. The chapter concludes with a section on speeding up object detection algorithms which is of importance to our work.
- **Chapter 3** introduces the datasets and evaluation methodologies used throughout this work. We motivate the choices for the datasets and discuss the data mining strategy for constructing our own TomTom rear-view car dataset. The final section of this chapter gives information on our evaluation methodology.
- Our baseline detector is introduced in **Chapter 4**. We describe the feature representation of integral channel features and discuss our learning framework. After a theoretical discussion we conclude this chapter with sharing a number of implementation details and summarizing the overall detection framework.
- Motivated by the observation that feature extraction and classification of subwindows contributes to a significant fraction of processing time per image, we suggest the implementation of so-called soft-cascades in **Chapter 5**. These methods focus on decreasing the average classification time per subwindow. More specifically we describe the WaldBoost algorithm for speeding up boosted classification.
- The next algorithmic improvement to our detection framework targets better multiscale handling. We show that construction of the image pyramid and performing

feature extraction and classification for every scale tends to be computationally expensive. Based on the scale invariance of image statistics in natural images, we describe an idea for multiscale object detection without image rescaling in **Chapter 6**.

- In **Chapter 7** we put all parts together by describing our overall detection framework. After doing so we perform a series of experiments both on the complete detector and on the individual components. The experiments on overall detection quality and speed benchmarks are most important and summarize the final performance.
- **Chapter 8** is the final chapter and contains the discussion and conclusion. We analyze our experimental results, mention notable outcomes and suggest ideas for future research. Also we include some reflection upon our own work and a make remark on the computer vision research community.
- There are two **Appendices** at the end of the work. The first one contains mathematical derivations and proof that were too long or not relevant to put in the main text. The second appendix contains a conference paper submitted to The Netherlands Conference on Computer Vision (NCCV). For readers not interested in reading the entire thesis we suggest studying the NCCV paper and Chapter 8.

2

Related Work

Object detection in images and video has received lots of attention in the computer vision and pattern recognition communities over recent years. Of all visual related questions we might want to solve on a computer, analyzing an image and *recognizing* all objects remains to be one of the most challenging ones. The image recognition problem can be broken down into a number of subproblems. For example, if we know what we are looking for, the problem is one of *object detection*. This task involves quickly scanning an input image and determining the locations where the target object is present. A different recognition task is the one of general *category recognition*, in this case the goal is to detect and categorize objects from extremely varied classes such as animals, furniture or cars [83]. In this work we focus on the problem of object detection: we know what we are looking in an image and want to detect the target objects as accurate and fast as possible.

Readers familiar with theory related to object detection may choose skip this chapter. The first section in this chapter includes a brief introduction to object detection. We describe common methodologies and the high-level organization of detection frameworks in practice (*i.e.* the detection pipeline). The remaining two sections of this chapter focus on related work. More specifically we introduce important feature representations, machine learning paradigms and object detection frameworks. We conclude with a brief survey of contributions to speeding up detection algorithms.

2.1 Brief Introduction to Object Detection

The object detection problem is formulated as follows. Assuming we have some object class of interest and an image to analyze, our goal is to develop a system which can detect instances of this object class in the image and return their locations. Object detectors typically return a list of bounding boxes and corresponding classification scores. For example when our target object is a car, the output of an object detector could look like the annotated image displayed in Figure 2.1. This example already points out some of the difficulties associated with object detection. The foremost difficulty in building a robust object detector is the amount of variation in images and videos. Important factors

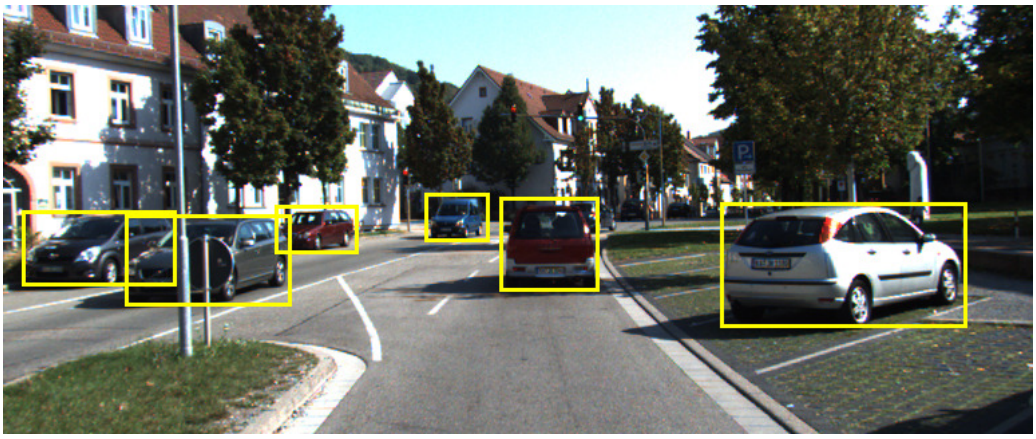


Figure 2.1: Example of object detection in a static image displaying a street scene. An object detection algorithm trained for finding cars should return the bounding boxes indicated in yellow each one also assigned a classification score. Original image was taken from [36].

that contribute to this variation include the object's position, distance or orientation with respect to the camera, large within-class variance in object classes, background clutter, color differences, changes in illumination and (partial) occlusions. The goal of object detection algorithms is to detect objects under all conditions and be as robust as possible to these variations. The quality measurement of an object detection algorithm is given in terms of detection accuracy and detection speed: we want to detect target objects under all circumstances as fast as possible.

We can consider object detectors as a combination of two key building blocks: a *feature extraction* algorithm that encodes image regions as descriptors, and a *classifier* that decides upon the class label of subregions in the image based on the feature representation [10]. In other words the first task is finding an informative representation for describing image regions, *i.e.* feature vectors or descriptors. After feature extraction, we apply a machine learning algorithm to classify image regions for determining the presence of certain objects. As is typical in machine learning problems there are two fundamental questions underlying this problem. The first question is: what is the most efficient way of describing each image region, *i.e.* what is the best feature representation? The second question is: given a feature representation, what learning algorithm is best capable of separating the object class from background class? We will continue to address these questions in a high-level discussion after which we elaborate on these by giving examples in the context of popular detection frameworks.

2.1.1 Feature Representation

The first question concerns the feature representation for object detection. For many problems the key to finding the solution lies in finding the right representation. In the case of object detection we need to transform raw, low-level input (*i.e.* pixel values) to a higher-level, more meaningful representation that is both highly informative and can be

computed with reasonably computational efficiency. The relevant question is: given an input image comprising of an enormous amount of input pixels, what are the best feature descriptors to encode the pixels into a meaningful representation for object detection? Over the recent years an extensive amount of research has contributed a vast number of image descriptors for object detection, keypoint extraction and tracking.

In computer vision there is a distinction between *global* and *local* image descriptors. Global descriptors describe the entire image and are generally not very robust as a minor change in part of the image may cause the descriptor to change significantly making it unsuitable for object detection. Local descriptors give a representation for smaller regions within an image and are more suitable for object detection, since a change in the image only modifies the descriptors in local neighborhood. A large collection of local image descriptor types is explored throughout the years, some of which are inspired by how objects are perceived by humans in the visual cortex of our brain [76]. Such image descriptors can be based on color information, corners or keypoints, edges, texture or combinations of these. The choice of feature representation largely depends on the problem that is confronted. For face detection one might find a different feature representation useful compared to the task of finding cars in an image. Selecting the right features plays a critical role in object detection – in general the most desirable property of visual features is their uniqueness so that objects can be easily distinguished in the feature space.

In addition to being useful to the specific problem, visual image features should to some extend be invariant to certain changes. Example include changes in translation, scaling, rotation, noise, illumination and geometric distortion [53, 2]. Features associated with images are called invariant if they are not affected by changes regarding the object view point. Not surprisingly no truly invariant features exist, but rather they are more or less robust to these changes. Designing features that retain these properties is a complex task which is often undertaken by many hours of trial and error. Robustness to changes in illumination are typically achieved by preprocessing images and feature normalization [10]. Invariance to translation and scale can be introduced by averaging (*pooling*) over pixel neighborhoods. In Section 2.2 we present an overview of popular and groundbreaking object detection frameworks and discuss their feature representation.

After determining what kind of features are best capable for the classification task at hand, another important question is how many features are sufficient. There is a minimum number of features that can effectively describe an object. An insufficient number of features may result in poor detections. On the other hand, too many features pose a problem for both training and detection phases. For example including more features typically increases both training and detection times substantially resulting in a very slow object detector. The behavior referred to *overfitting* also poses a serious problem when adding more features or parameters to the predictive model: adding more parameters can produce excellent results on the training set but fails to generalize for unseen test examples. Reducing the effect of overfitting can be achieved by increasing the number of training samples. The difficulties that can arise in feature spaces of many dimensions are sometimes referred to as the *curse of dimensionality* [6].

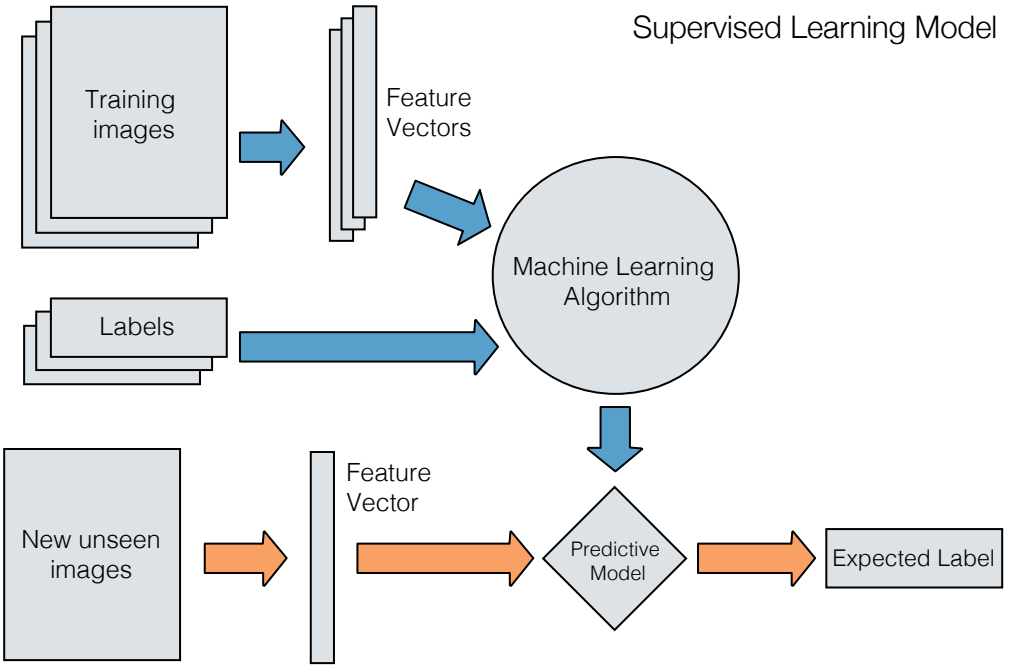


Figure 2.2: Training and classification process for supervised learning [63].

A recent survey paper [5] on object detection shows that over the last couple of years the most significant improvements to detection quality have been made by developing better feature representations. Designing richer feature representations that can be extracted efficiently seems to be more beneficial than improving the machine learning algorithms for classification. This can be explained by the fact that given good feature vectors, supervised learning algorithms are already excellent in optimizing the decision boundary separating the object categories. Furthermore these classification functions can be evaluated efficiently at detection time and typically require less computation time than feature extraction. Therefore putting focus on decreasing feature extraction time will be more profitable for speeding up object detection. Unfortunately the process of designing new image descriptors largely remains to be a process of trial and error – the scientific community faces the knowledge gap of attaining a more profound understanding of what makes a certain feature representation good and how to design better ones.

2.1.2 Learning Classification Functions

The next step in building a predictive model for object detection is training our classifier using some machine learning algorithm. This process includes several steps indicated in Figure 2.2. The first step is to obtain a training set of annotated objects that can be used for learning. In a binary classification task of object detection the training set typically consists of a large set of images of the target object and a large set of non-object examples (background patches). Given these two image sets for which the class labels are available we extract feature vectors (descriptors) for all training samples. These feature vectors are then fed into a machine learning algorithm which attempts to build a predictive model

based on the class labels and feature vectors of objects in the training set. In statistical classification problems the learning algorithm optimizes the decision boundary which is the hypersurface that partitions the underlying feature vector space into distinct regions for each object category. For a two-class classification problem the classifier will assign all the points on one side of the decision boundary label +1 while the point on the other side will be assigned label −1. The model learned by the machine learning algorithm should have excellent prediction capability, meaning that we can accurately predict the class label of new unseen objects.

Machine learning techniques such as support vector machines [40, 75], boosting [73, 43] and neural networks [46, 47] have become the most popular learning algorithms for object detection. The success of these learning frameworks in object detection is mostly due to their ability to automatically select the relevant image descriptors from large feature pools, their excellent performance in high-dimensional feature spaces and relative ease of use. The object detector we discuss in Chapter 4 takes advantage of boosting for both feature selection and classification.

For object detection in full images, generally we are interested in classifying smaller regions (subwindows). Therefore the object detector is trained on a fixed model size. All training examples are rescaled to the same dimensions and the features descriptors are computed using the pixels at this scale. Sometimes the object detector of model trained at fixed size is referred to as a (rigid) template. Model size and aspect ratio should be chosen to be appropriate for the object class, *i.e.* for a car detector the template could be of square size while portrait orientation is better for pedestrian templates. The detector is only able to detect objects of this particular size but as we will see in the next section fortunately there are ways for performing multiscale object detection in full images.

2.1.3 Object Detection in Full Images

For detecting objects at all positions in an image a commonly adopted detection and localization approach is the *sliding-window* technique. Moving across the image the sliding-window scans the image by feeding the descriptors of each subwindow to the classifier. The specified sliding-window step size at which the detector scans across the image is an important parameter influencing both speed and detection rate. Subwindows classified as positive are saved as object detections characterized by the position in the image and classifier score. Since the total number of possible subwindows is vast, the sliding-window is generally applied with a certain stride or step size, *e.g.* 4 or 8 pixels. Consider the scenario where we want to perform object detection in an image of 800×600, a trained template of 64×64 and a step size of 4 pixels. This already results in 24.656 subwindows that need to be processed by the classifier for a single image over one scale.

Sliding-windows are designed for object detection at all locations in an image, however in practice objects generally do not appear at the same size and aspect ratio as the trained template. Since the sliding template is trained at a fixed size it can only detect objects of

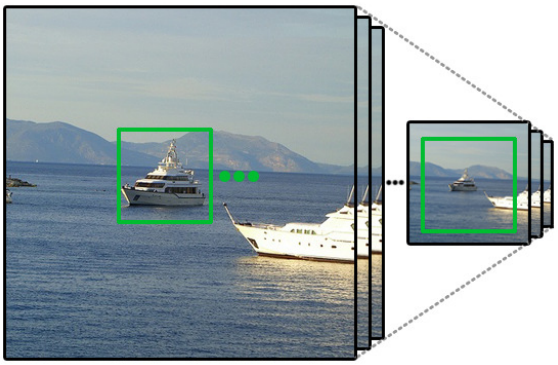


Figure 2.3: Image pyramid for multiscale detections using a sliding-window. The image pyramid can contain more than 30 levels which requires significant computational resources.

this particular size. To overcome this problem the traditional detection pipeline includes construction of the *image pyramid* by rescaling the input image several times. Aside from actual feature extraction and classification, the construction of the image pyramid is a computationally expensive task and can take more than a second on a modern desktop computer. After constructing the image pyramid, multiscale object detections at all locations and scales can be achieved by applying the sliding-window to each level of the pyramid. This process is illustrated in Figure 2.3. Not surprisingly the the number of subwindows proposed to the classifier rapidly grows with the multiscale analysis over all levels in the image pyramid.

Classification of all subwindows encountered by the sliding-window over all scales in the pyramid typically results in a list of object proposals comprising of a bounding box and detection score. Since detections are made over a large number of adjacent scales and locations each object is typically detected multiple times at slightly different size and position. The last step in the detection process is to group nearby detections so that every object is only detected once, *i.e.* non-maxima suppression (NMS). Simple NMS algorithms are straightforward and group together overlapping detections and only maintain the detection with the highest detection score. A more advanced method for grouping nearby detections is the mean-shift algorithm described in [10]. The high-level detection pipeline as described in this subsection is summarized in Figure 2.4.

Figure 2.4: High-level overview of the *detection* pipeline for object detectors.

2.2 Popular Object Detection Frameworks

The introduction of new local and semi-local features arguably is the most significant contribution to advancements in object recognition systems over the last decade. Top

performing features are typically to some extend invariant to illumination changes and small deformations. A large amount of recent object detection frameworks use either Haar-like or gradient orientation based features. Furthermore, recent advances in deep learning and more specifically convolutional neural networks have received great interest from the computer vision community. While numerous widely used object detection frameworks are available in the literature, we select the three detection frameworks that are most important in our opinion. We study feature representation and learning paradigm. Our discussion begins with the work by Viola and Jones [88] on face detection after which we continue to describe the pedestrian detector by Dalal and Triggs [11] that introduces the widely used histogram of oriented gradient features. Finally we conclude the section with a brief introduction to convolutional neural networks by the collaborative works of Hinton and LeCun [46, 48]. The works are discussed in chronological order of their first introduction to object detection. For our own work we build upon integral channel features which can be considered as a combination of Haar and HOG features described in the upcoming subsections. Since channel features are the feature representation of our detector we introduce these in Chapter 4.

2.2.1 Viola-Jones Framework

Viola and Jones [88] proposed one of the first object detection algorithms that was useful in practice due to its high speed and detection accuracy for face detection. The authors choose a simple feature representation based on Haar features capturing coarse differences in illumination between adjacent image regions. Major contribution of their work is the use of the *boosting* machine learning paradigm for feature selection and learning classification functions. Additionally the VJ detection framework was the first to include *integral images* for rapidly summing pixel regions necessary for feature extraction. While originally described in context of face detection, the framework has been extended many times and is also one of the core ideas behind the current work although we choose a different feature representation.

As mentioned, VJ use simple Haar features (wavelets) which are originally introduced by Papageorgiou *et al.* [60]. These features consist of two, three or four adjacent pixel regions from which a feature value is computed by taking the difference in pixel sums between those regions. More specifically, Figure 2.5 illustrates the three different feature types used by VJ. Black and white regions indicate the positive and negative contributions to the feature value respectively. Extensively summing pixels over these regions can become a computationally expensive process, hence the authors suggest the idea of using integral images from which summations can be computed by simple pixel lookups. This is an idea that also finds it way into our object detector, hence we describe it Section 4.1.2.

Rather than manually selecting the most informative Haar features for the classification task, the authors propose automatic feature selection using the boosting framework proposed by Freund and Shapire [31]. The training process starts with the creation of an enormous candidate feature pool consisting of randomly generated Haar features only

constrained in width and height by the model size. The boosting algorithm iteratively selects the best features from the initial pool. With "best" is meant the feature that has the ability to divide the positive and negative training samples into distinct classes with the lowest number of (weighted) misclassifications. Although we use a different feature representation, this feature selection process is also the basis of our detection framework.

For classification rather than using a single complex classifier, VJ adopt a cascade-like structure of simpler classifiers that are becoming more complex while progressing through the chain. Each stage in the chain is designed to reject as many of the remaining negative cases as possible while still retaining all positive samples [89]. This structure efficiently rejects non-promising subwindows in the image reducing overall classification time.

There are two main strengths of using this feature representation in a boosting framework. First and foremost the computational efficiency comprised by the features. Since features are extracted using pixel lookups from integral images the cost of feature extraction is very low compared to other types of features. However the main computational advantage is the ability to perform multiscale analysis without rescaling the input image. Learned features can be upscaled or downscaled by simply adjusting the feature regions and consequently modifying the detection threshold. As we will see this is an incredibly favorable property since it allows for multiscale object detection without explicitly rescaling the input image. This makes the VJ detection framework to be one of the fastest detectors available to this date. Chapter 6 of our work is devoted to the idea of rescaling the features not the images. The second strength of the feature representation by VJ is the easy interpretation of the features that are learned. For face detection it is observed [89] that Haar features focusing on the contrast differences above and below the eyes and left/right of the nose are the most informative. This observation is not surprising if we think about how we humans perceive people's faces.

In comparison to other object detectors the VJ algorithm also has a number of disadvantages. For example due the larger features the localization accuracy is lower and relies on the non-maxima suppression of nearby detections. But its main drawbacks are performance degradation with object rotation and the sensitivity to changes in illumination. However the novel ideas on feature selection using boosting, detection cascades, integral images and the scale invariance of Haar features remain popular components of modern object detectors.

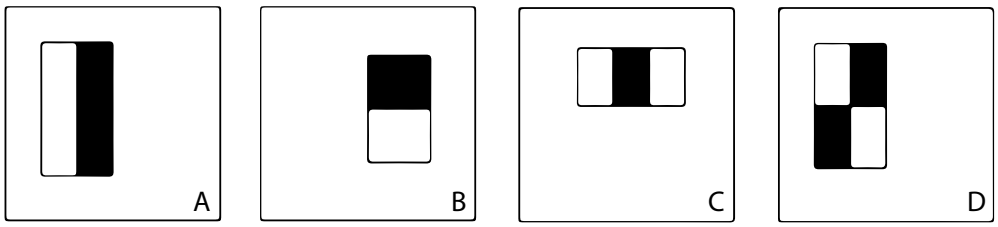


Figure 2.5: Rectangular Haar features used by Viola and Jones [88]. Feature values are computed as the difference between pixel sums of the black and white regions.

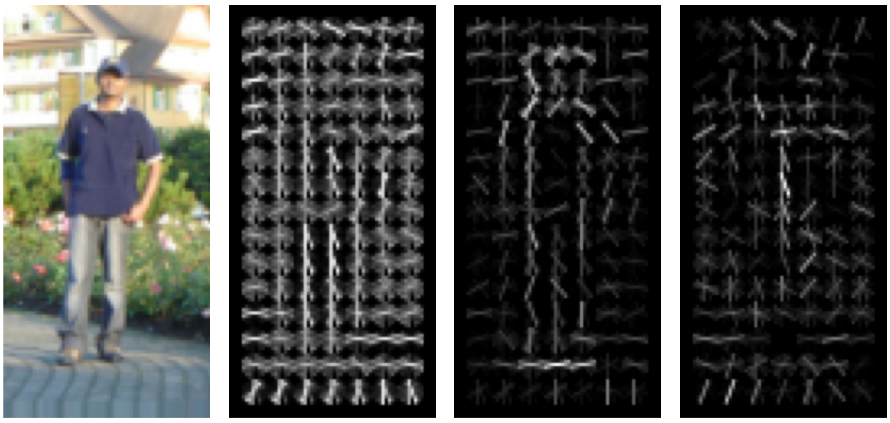
2.2.2 Histograms of Orientated Gradients

The more recent detector proposed by Dalal and Triggs [11] is also highly popular and adopts a completely different approach compared the VJ detection framework. Their work focuses on improving the feature representation with the introduction of *histograms of oriented gradient* (HOG) descriptors inspired by the work on SIFT keypoint descriptors by Lowe [53]. Their learning framework consists of linear support vector machines (SVMs) for speed and simplicity – the focus of the paper is more on feature representation than on learning classification functions. For detection the trained classifier of fixed model size, also referred to as rigid template, is applied as sliding-window at all positions and scales of an image. The original ideas are proposed in the context of pedestrian detection but since then the ideas are widely adopted and are at the foundation of many object detectors.

The essential thought behind HOG descriptors is that local object appearance and shape within images can be described by the distribution of intensity gradients or edge directions. For computing descriptors from an image, the image is divided into a large number of small connected cells. Each HOG cell is of fixed size and accumulates magnitude-weighted votes for gradient at particular orientations, *i.e.* the gradient orientation histogram. The original idea of composing gradient orientation histograms for small image regions is due to Lowe in his work on SIFT descriptors [53]. However unlike SIFT features, which are designed as keypoint descriptors, HOG features are contrast-normalized over a larger image region called a block. The normalization helps to create descriptors that are to some extend robust to changes in illumination, shadowing and background clutter.

The final step in the training process of the Dalal-Triggs object detection framework is training the classifier by feeding the HOG features into SVMs for building a predictive model using supervised learning. The choice of using SVMs as binary classifiers is motivated by its excellent performance given high-dimensional feature vectors and its ability to quickly optimize the decision functions in the training process to find the most discriminative features that can also be evaluated efficiently at detection time. The optimization of the decision boundary for SVMs is described by Schölkopf *et al.* [75]. Figure 2.6 displays HOG features extracted for a pedestrian and also shows the feature weights assigned by the SVMs after training. The positive weights in subfigure 2.6c clearly shows that the regions around the head, feet and torso are most discriminant for pedestrians and thus receive more weight in the trained classifier.

Today one of the most popular object detectors is the work on *deformable part models* (DPM) by Felzenswalb *et al.* [24]. The DPM detector is a natural extension to the work on HOG features and includes the idea of pictorial structures to represent objects by a collection of parts arranged in deformable configurations. This results in an object detector that is very robust to deformations and occlusion [25]. Each part characterized by HOG features captures local appearance properties of an object while the deformable configuration is constructed by spring-like connections between certain pairs of parts [23]. In other words, the general idea is that an object is represented by a root model and multiple parts as



(a) **(b)** **(c)** **(d)**

Figure 2.6: HOG features and their weights assigned by linear SVM training. **(a)** Example image from the training set. In **(b)** a visualization of the HOG descriptors is shown indicating the strength of orientations within each cell. Figures **(c)** and **(d)** display positive and negative weights assigned by the linear SVM respectively. Image was taken from [10].

illustrated in Figure 2.7. The part-based model is trained using *Latent SVMs* in which the input-output relationship is not only characterized by feature vectors and class labels, but also depends on a set of hidden latent variables [92].

The idea of building a model from parts as introduced with DPM is an concept that has become very popular and dozens of variants have been explored. The combination of a root filter surrounded by a number of parts make these detectors robust for dealing with occlusions. However the model complexity that achieves this robustness is accompanied with high demands on computational resources, therefore the detector is very slow in its original form. The original paper reports average detection times 2 seconds per image running on a powerful machine with 8 CPU cores and a parallelized implementation [24]. Fortunately a significant amount of speed-up proposals for DPM have emerged over the recent years, some of which will be discussed in the next section. Since DPMs are known to achieve state of the art detection results and are relatively easy to train, it is common in object detection literature to report DPM detection results as comparison. We also benchmark our detector against the DPM detector for comparison.

2.2.3 Convolutional Neural Networks

Even though our work shares almost no similarities with the recent works on convolutional neural networks for object detection we feel that a brief explanation of the ideas is appropriate due to its great interest from the computer vision community. Besides this subsection explaining the high-level ideas we will not refer to these recent works on advances in object detection using neural networks.

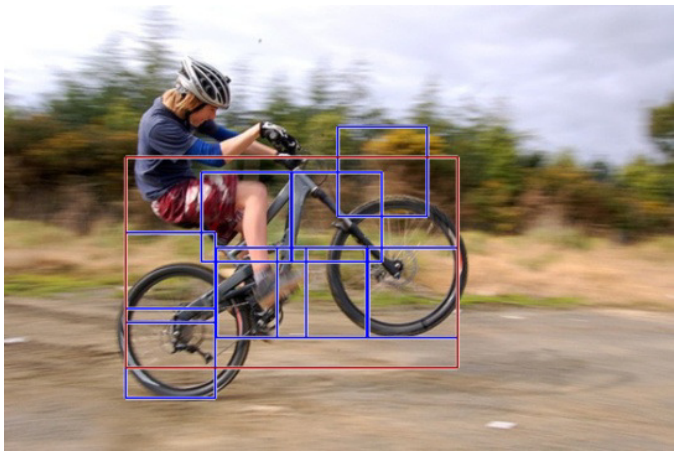


Figure 2.7: Detection of the DPM detector using a bicycle model. Red rectangle denotes the response by the root filter while the blue rectangles indicate the part locations. Notice that the components can deform to detect a "wheelie" by the bicycle. Image was taken from [24].

All the previously described object detectors share a common factor which is that features to some extend require manual engineering to obtain meaningful descriptors from raw pixel data. Conventional machine learning techniques are limited in their ability to process data in their raw form. Deep learning is a representation learning method with multiple levels of representation complexity, from low-level image features (*e.g.* edges or corners) to high-level features such as texture or parts of objects [49]. Starting with raw input data, neural networks compose simple but non-linear functions to transform the representation into slightly more abstract levels capable of describing structure of the data. Given the fact that deep learning networks are excellent in finding meaningful structures in high dimensional data they are also becoming increasingly popular for object detection, more specifically in the form of *convolutional neural networks*.

Convolutional neural networks (CNNs) [46, 82] are designed for processing data that comes in the shape of multiple high dimensional arrays, for example the pixels of an image. Four key components characterize the performance of CNNs: local connections, shared weights, pooling and the use of many layers [49]. The architecture of CNNs is given by a series of stages of convolution layers, pooling layers and fully connected layers. More traditional neural networks model a neuron's output as a `sigmoid` or `tanh` function, however in terms of training time these are much slower than the non-linear function $f(x) = \max(0, x)$ where x represents the input of the neuron. Deep convolutional networks use the so-called Rectified Linear Units (ReLUs) as activation function to train several times faster than conventional activation functions. Precise architecture can be highly complex and specific to the classification task at hand. Hinton *et al.* [46] describe their deep convolutional network for classification of 1.000 object categories comprised in the ImageNet dataset. Their network consists of 60 million parameters trained on 1.2 million images taking over 6 days on two powerful GPUs.

Although most papers on CNNs for object detection mention the training time, reports on the evaluation time of input images are limited. Authors in [38] apply CNNs for 5-category object detection on the PASCAL VOC dataset and report detection time of 10 seconds per image running on a powerful desktop with GPU acceleration. It seems like CNNs are currently more focused towards multi-class classification on very challenging datasets such as ImageNet or CIFAR, rather than focusing on high-speed object detection.

2.3 Speed Improvements for Object Detection

The emphasis of our work is on improving detection speed and computational efficiency. Providing an exhaustive overview of publications on high-speed object detection is beyond the scope of this thesis, however we will summarize a number of techniques that are related to our work and we believe are important to achieving the highest possible detection speeds. Speed improvements for existing object detection frameworks can roughly be categorized into (1) improving the feature representation, (2) speeding up the classification of subwindows, (3) focusing on fast multiscale detections and (4) computational improvements. Below we summarize important contributions to these topics in relation to sliding-window based object detector.

Feature Representation. Over the last couple of years there tends to be more focus on improving features rather than on improving classification [5]. Meaningful and efficiently extractable features are crucial to good detection rates. The first generation of object detectors frequently employed Haar features. While fast to compute using integral images, the popularity of Haar features decreased mainly due to the introduction of histograms of oriented gradient (HOG) features. A recent survey paper on pedestrian detection [4] shows that many of the high-performing detectors use a HOG-like feature representation. It seems like gradient orientation-based features are highly efficient in encoding edge, corner, key-point and local-shape information, while being reasonably fast to extract. Although HOG features with SVM classification [11] are not particularly slow, the specific normalization scheme and high dimensional feature vectors hinder computational speedups. Our detector uses channel features which combine the computational efficiency of Haar features with the information richness of HOG features.

Classification Speedups. Inspired by the boosting cascade proposed by Viola and Jones, many research has been devoted to so-called *soft-cascades* [79, 93, 91] for speeding up the detection process. These works focus on pruning negative subwindows as early as possible in the detection process. During the training process in addition to selecting the best features and thresholds also rejection thresholds are learned. When at evaluation time the cascade score drops below this threshold the subwindow is immediately pruned and labeled as non-object window. Typically non-promising windows are rejected early in the cascade, resulting in a significant speedup when the number of windows to evaluate is large as with full image detections.

Multiscale Models. Typically for detecting objects of various size, the input images are rescaled a number of times (*e.g.* 30 scales) and features are extracted for each scale. Computing the image pyramid and extracting features at every scale requires a significant amount of computational resources. To overcome this computational burden some authors propose interpolation of features over scales to reduce the number of scales in the image pyramid. Dollár *et al.* [13] introduce a method for approximating various feature types over adjacent scales using the so-called power law for feature scaling. Since then this technique has been employed with great success for pedestrian detection [3] and general object detection using deformable part models by Saghedi *et al.* [72]. This technique is also an important aspect of our detector and dramatically reduces computation time spent on image rescaling and feature extraction (Chapter 6).

Computational Speedups. In a different line of work, several improvements focusing on computational speedups have been proposed. Zhu *et al.* [94] show that HOG features can be rapidly computed using integral images. Furthermore various GPU implementations for speeding up computer vision algorithms are described [65], and a number of low-level operations are optimized with vector operations, multiple cores and CPU cache management [72].

The work by Beneson *et al.* [3] on performing high-speed object detection without image rescaling shares the most overlap with our current work. Similar to our work the authors propose a very fast object detector for pedestrian detection using integral channel features, boosting and an approximation scheme for multiscale detections. Core component for reaching high detection speed in their detection framework is the ground plane estimation technique for reducing the search space. The authors report detection speeds of 100 fps, using the ground plane estimation technique and stereo-camera. Using the stereo-camera setup and the ground plane estimation technique the authors are able to reduce the search space to $640 \times 60 \cdot 10$ scales, hence this allows them to run the detector at such impressive frame rates. Our work is different since we do not consider ground plane estimation but rather focus on speeding up the classification of subwindows.

3

Datasets and Methodology

In this relatively brief chapter we discuss the datasets that are used throughout this research. Introducing our datasets at this stage allows us to put our theory in the context of these datasets and discuss some preliminary results in the following chapters. In addition to simply listing our datasets we also present some background information, data mining methodology and the specifications of our image collections. The final section is different and introduces the evaluation measures used for determining the detection quality of the object detectors on our datasets.

3.1 Datasets

Typically one of the more challenging aspects of solving a machine learning problem is the collection of labeled data for training classifiers. For supervised learning, datasets of thousands or millions of annotated training examples are common. To meet this demand for training samples, there is an ever growing number of datasets available. In the context of computer vision, datasets are often large image sets from which feature descriptors can be extracted. These datasets typically contain two large image sets: positive training samples exhibiting the object of interest and negative training samples displaying background patches. Additionally the data is often divided into a train and test set; the training group is purely used for training whereas the performance of the trained classifier is determined on the test set. Cross-validation and leave-one-out methods can be employed for obtaining accurate detection statistics when the dataset is not divided into two subsets [84]. Datasets are typically put together by hard work including data collection, manual annotation and adding contextual information. Fortunately in recent years crowd-sourcing platforms like Amazon Mechanical Turk have emerged for outsourcing the labeling process to people around the world.

Computer vision datasets, and more specifically for object detection, often seem to focus on the task of *pedestrian detection*. There is a large number of pedestrian datasets available making pedestrian detection one of the benchmark problems in object detection. Popular datasets include the INRIA Person dataset [10], Caltech pedestrian dataset [19] and the

Daimler benchmark [35]. However as we have defined in the scope of our project, we focus on the task of *car detection*. Unfortunately the number of available datasets for this purpose seems to be significantly smaller than pedestrian datasets. To our knowledge the only serious candidates are the KITTI Vision Benchmark Suite [36] and TME Motorway dataset [8]. We would also make a mention of the PASCAL VOC dataset [22] which is frequently used in context of general object detection and we use for mining of negative training examples.

3.1.1 KITTI Vision Benchmark Suite

We begin with a short discussion on a dataset that did not make it into our final work; we explain our attempt to use this dataset for training and why it did not produce satisfying results. When searching for good datasets with focus on automotive applications such as car and pedestrian recognition, one will certainly come across the KITTI Vision Benchmark Suite [36]. Taking advantage of their autonomous driving platform, the researchers from the university of Karlsruhe have developed a series of datasets for benchmarking optical flow, stereo vision, visual odometry, SLAM and object detection. The object detection dataset consists of 7250 panorama images at 1242×375 resolution which can be used for training (see Figure 3.1). We praise the authors for the convenient dataset formats, the enormous amount of information on all labeled objects and the high resolution images. Using the annotation files we were able to set a filter on the car orientation, distance and object type to extract a total of 1500 rear-view car images.

Figure 3.1: Example image from the KITTI Vision Benchmark Suite [36]

During the early phase of this project we used the KITTI Vision Benchmark Suite for training and evaluating our object detector. However after performing experiments we found the level of variation in training examples insignificant. Our main point of criticism is the low amount of variation in the images from which training samples are extracted. We performed a simple cluster analysis the set of training examples. By clustering the 1500 positive car samples contained in the KITTI dataset we made the observation that identical cars appear several times in the dataset. This is illustrated in Figure 3.2, which gives an impression of the within-cluster variance. Some of the clusters all contain the same car, for example there is one cluster of more than 100 samples exclusively containing

Figure 3.2: Cluster analysis of KITTI Vision Benchmark Suite. We performed *k-means* clustering with $k = 10$ to show that some of the clusters with $n > 50$ exclusively contain multiple copies of the same car.

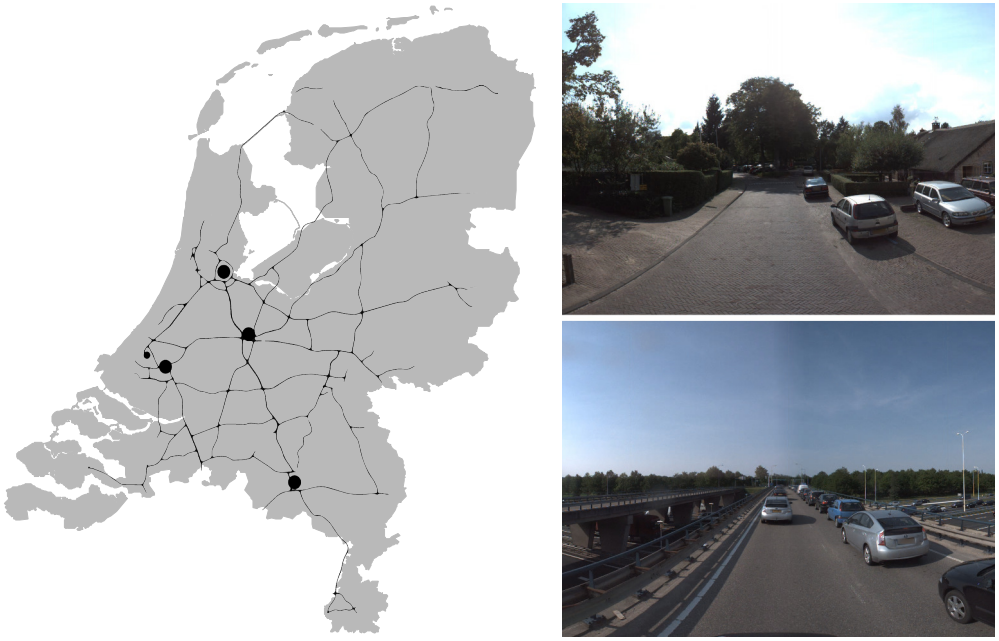
the same Coca-Cola truck. Apparently the dataset was constructed by driving around and labeling the cars that are visible on camera. The disadvantage of this process is reflected by the cluster analysis. This introduces a bias in the sense that our learning algorithm tends to assign more weight to the shape and color of a Coca-Cola truck. Although including the appearance of the truck in our training set is important for robustness, including it multiple times will not be beneficial to detection rates. One could simply take one training example from each of the clusters, however this leaves us with a very small training set.

Based on the cluster analysis we concluded that more samples, capturing a larger amount of variation in car appearance, are required. We committed ourselves to construct a new dataset with the purpose of training our car detector. This decision is motivated by the fact that while staying at TomTom, we are fortunate to have access to a large amount of image data captured by mobile mapping vans driving around the world. We made good use of this vast image database by performing semi-automatic data mining to construct a novel dataset specifically for this work. In the next section we describe the data mining process and the characteristics of our new dataset.

3.1.2 TomTom Dataset

TomTom has its own fleet of mobile mapping (MoMa) vehicles driving around all over the world. These cars are equipped with high-resolution panorama cameras that capture full 360-degrees around the car. This data is similar to Google Street View images. For this work we had access to the entire image database, giving us the opportunity to perform data mining for the construction of our dataset. Unfortunately the objects of interests are not annotated in the database, so we designed a custom tool for semi-automatic data mining.

Querying the MoMa dataset is performed by sampling GPS coordinates by sending these to an API. The back-end returns the nearest location visited by a mapping vehicle. This information can be used for requesting panorama images captured at that specified location. Our initial sampling strategy was selecting uniform at random a GPS location



(a) GPS sampling (highways and cities)

(b) Examples images TomTom dataset

Figure 3.3: Data mining of the MoMa database. GPS positions are randomly sampled on highways and in cities of The Netherlands. The black overlay on the map indicates the region of GPS positions are sampled at random.

within the bounding box The Netherlands. We quickly realized that the majority of images returned by this sampling strategy did not contain cars. This is not surprising since the majority of images is captured on empty roads in rural areas. Therefore we improved our method by only sampling GPS coordinates within cities and on highways. This turned out to be an efficient strategy for data mining, although we recognize the selection bias that comes with our sampling strategy. Figure 3.3 indicated the highway and city regions that were sampled and two example images comprised in the dataset. We allowed the data mining script to run during a full night until we collected a large amount of images all containing at least one car.

Panoramas are returned as single image blocks which need to be stitched together manually. After obtaining the full images we annotate the bounding or cars and vars. To label bounding boxes in the panoramas we choose a semi-automatic method. We wrote a simple tool for proposing bounding boxes based on the output of a pre-trained deformable part-based model detector of the OpenCV repository (see Section 2.2.2). This object detector requires more than 2 seconds of processing per image however the time constraint is not an issue for data mining. The DPM detector proposes bounding boxes which can be added to the dataset with simple keypresses. Additionally bounding boxes not proposed by the detector are manually included. We emphasize that the DPM detector was only used for proposing bounding boxes, all observations are manually checked for correctness.

In this semi-automatic process we constructed our dataset consisting of 2366 positive training samples all containing rear-view of cars and vans. All extracted samples are rescaled to 80×80 pixels. The dataset is randomly divided into a train and test set. The train set contains 2000 positive samples while the test set uses the remaining 366 examples. These subsets are kept constant through the research, *i.e.* after initial sampling into the two subsets they remain fixed. We also extracted a large set of negative training samples from the TomTom data containing background image patches that do not contain cars, vans or trucks.

Within the set of positive examples there is a large amount of variation in color, shape, car brand and viewing distance. We acknowledge the fact that the images in our dataset are captured exclusively during daytime and with excellent weather conditions because MoMa cars only drive under these circumstances. Additional bias in our dataset is introduced by the roof mounted camera at fixed height on the mobile mapping cars. We recognize the imperfection of our dataset and emphasize that there is room for improvement, foremost by adding training samples captured under different weather conditions and using different camera height and hardware to increase the data variety. We are also interested in the importance of sample alignment, *i.e.* how precise the cars are in the center of the crops, however an in-depth analysis of the influence on detector performance is beyond the scope of this project.

3.1.3 TME Motorway Dataset

The Toyota Motor Europe (TME) Motorway dataset is a benchmark dataset composed for tracking and object detection in autonomous driving applications [8]. It consists of more than 20 video sequences accounting for a total of approximately 30 minutes. All sequences are recorded using a stereo-mounted camera below the windshield on highways in North Italy for both daylight and sunset conditions. Camera resolution is 1024×768 and image data is stored as PNG files.

Images in the dataset are annotated in a semi-automatic way using a laser scanner. The authors have developed an algorithm for car detection from 3D point cloud data which is utilized for writing annotation files for each video frame. Since the sampling frequency of the laser scanner (12 Hz) is significantly lower than the camera frame rate (20 Hz), the authors track detections over multiple frames and use interpolation to build a consistent set of "ground truths". Based on the width of the bounding box each observation is categorized as either car of truck [8].

Unfortunately the dataset has a number of disadvantages as the authors themselves also report. First and foremost the interpolation between consecutive video frames can result in bounding boxes that are considerably off from the ground truth in the image. Also the reliability of the generated data decays for observations beyond 60-70 meters due to limitations of the laser scanner. The format of annotation files is also not as convenient as we hoped for: bounding boxes are specified as *world* coordinates which should be

(a) Positive Samples

(b) Negative Samples (Background)

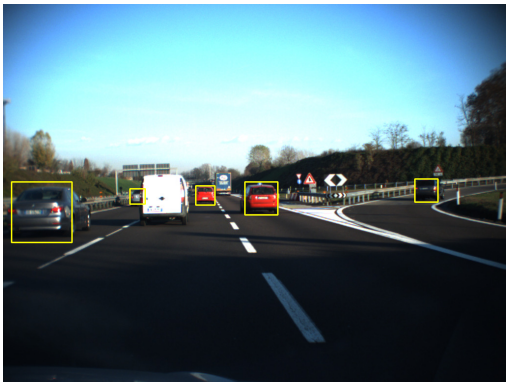
Figure 3.4: Training samples from TomTom dataset. Note the partial occlusion in one of the positive samples. Our positive training dataset contains 2000 positive and 2000 negative examples.

converted to *screen* coordinates using the camera calibration matrix. When analyzing the ground truth bounding boxes we confirm the limitations noted by the authors. The accuracy of bounding boxes is clearly not perfect, in particular for cars at some distance. We also report inconsistency with regard to occlusion. In some cases partially occluded cars are included as ground truth while this is not true for other cases with a similar occlusion level. An example of this inconsistency is visible in Figure 3.5 (right) where two cars are not annotated due to occlusion and distance. To overcome the problems with this dataset we have decided to manually fix the annotations of a subset of video frames.

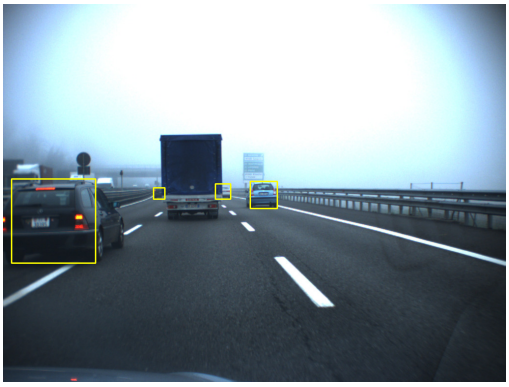
For all the available video sequences we extracted every 50^{th} frame and annotated all rear-view cars and vans. Vehicles with more than 50% occlusion or truncation (out of screen) are not considered as ground truth objects. Since our detector is trained at square size, we also rescale annotations to square dimensions. Using this annotation strategy we obtain an evaluation dataset of 465 video frames contributing to a total of 1298 labeled objects. Due to various lighting conditions, wide scale variety and occlusions this dataset is not an easy classification task. Example video frames with annotations are displayed in Figure 3.5. An interesting aspect of our evaluation data is the size distribution of cars and vans in our dataset. Figure 3.6 displays the size distribution of the bounding boxes in the TME dataset.

(a) Normal conditions

(b) Difficult lighting conditions



(c) Occlusions



(d) Cars in the distance

Figure 3.5: Four video frames from the TME dataset that used for detector evaluation. Only cars and vans with a rear window are annotated, trucks are excluded. All annotations are scaled to square size. [8].

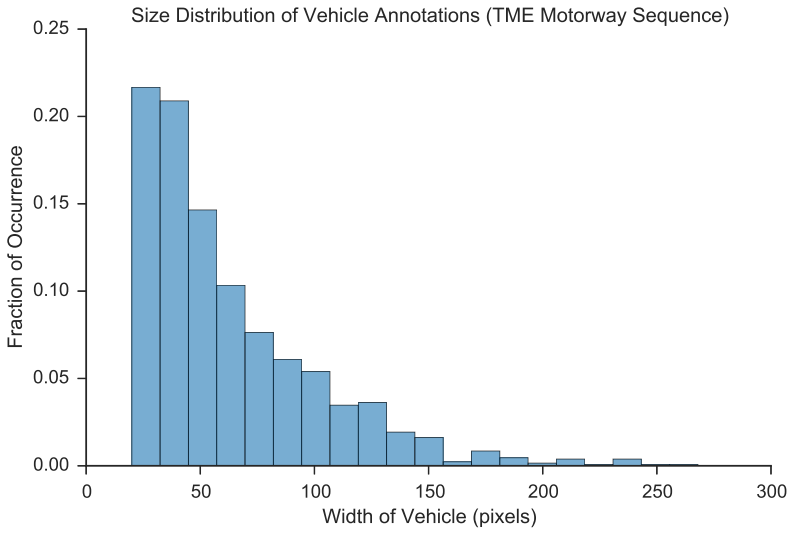


Figure 3.6: Size distribution of the 1298 annotated vehicles in the 465 video frames from the TME Motorway Sequence. Note that the majority of bounding boxes is small, *i.e.* the number of cars in the distance is significant.

3.1.4 Overview of Datasets

Since we use a number of datasets with different purposes, in this section we summarize the datasets used throughout this work. While not discussed explicitly we also extract negative training samples from the PASCAL VOC [22] dataset to increase the variety in our negative training set. Negative samples from the TomTom dataset might contain a limited amount of color and texture variety since images captured by mobile mapping vehicles include significant amounts of road, sky and trees.

In addition to static images we also collect two video files of resolution 640×480. These video sequences are employed for speed benchmarking our detector. Video streams are far more efficient in encoding video image data compared to simply reading PNG files from disk and encoding these individually. Furthermore performing our benchmarks using video streams is more in agreement with real world performance since image data recording by a camera is typically encoded as video stream. Also this gives us the opportunity to study the performance using images with possible compression artifacts introduced by the video codecs. The complete overview of all datasets is given in Table 3.1.

Table 3.1: Overview of all datasets used through this research. Training of our detectors is performed on our TomTom dataset, and we additionally include a set of negative training images from the PASCAL VOC dataset. The remaining table entries are utilized purely for detection evaluation.

Name	Purpose	Images	Resolution	Type
`TomTom positive`	detector training	2000	80×80	cropped images
`TomTom negative`	detector training	2000	80×80	cropped images
`PASCAL negative`	detector training	2000	80×80	cropped images
`TomTom positive`	evaluation cropped images	366	80×80	cropped images
`TomTom negative`	evaluation cropped images	366	80×80	cropped images
`TME dataset`	evaluation full images	465	1024×768	full images
`TomTom sequences`	speed benchmarks	n.a.	640×480	video (mpeg4)

3.2 Evaluation Methodology

For evaluation of our object detector we make a distinction between two different cases. First we evaluate detection results on cropped images of cars and background samples to determine the feature quality and performance of the learning algorithm without concerning about multiscale analysis and sliding window difficulties. The second evaluation method covers these two aspects by focusing on the localization of the objects in full images. Obviously this method results in a more realistic view of our detector since it requires the full detection framework including sliding window technique, multiscale analysis and handling large images. Determining classifier accuracy for cropped images can simply be done by requesting a label of the 80×80 cropped car or background samples. For evaluation of full images we need to include additional steps: matching detections to ground truth boxes, collect false positives and output the detection quality. In the last subsection of this chapter we describe what evaluation metrics we use for assigning a number of the concept of "detector quality".

In this section our attention is pointed to the *accuracy* evaluation of object detectors. Many different statistical and machine learning metrics are available for evaluating performance of classifiers. These include ROC curves, F-scores, precision-recall and false positives per-image [28]. During this research we choose to report precision-recall statistics for detector performance on cropped image, since they are typically utilized in machine learning literature and are revealing with regard to the sensitivity of the detector. In the context of object detection we compare the results of our detector to a given list of ground truth bounding boxes. First we need a measure for determining what detections are correct, *i.e.* what detections can be matched to a ground truth bounding box. After deciding upon the (in)correctness of each detection we propose precision-recall curves for evaluating detector performance.

3.2.1 Overlap Criterion

The first question to answer is when a detection is considered correct. Literature on object detection commonly uses the *overlap criterion* introduced for the PASCAL VOC challenge formulated by Everingham *et al.* [22]. As noted, for the detection task the detector returns a list of bounding boxes with associated confidence (score). Detections are assigned to ground truth objects and judged to be true/false positive by calculating the bounding box overlap. To be considered a correct detection, the overlap ratio a_0 between the predicted bounding box B_p and the ground truth bounding box B_{gt} must exceed 0.5 (50%) according to [22]. The **PASCAL VOC overlap criterion** is defined as the *intersection over union* for a detection with a ground truth annotation:

$$a_0 = \frac{\text{area}\ (B_p \cap B_{gt})}{\text{area}\ (B_p \cup B_{gt})} > 0.5. \tag{3.1}$$

Table 3.2: Terminology of the four observation types.

Symbol	Definition	Meaning
TP	True Positives	Correct detection
TN	True Negative	Correct rejection
FP	False Positive	False alarm (Type I error)
FN	False Negative	Missed detection (Type II error)

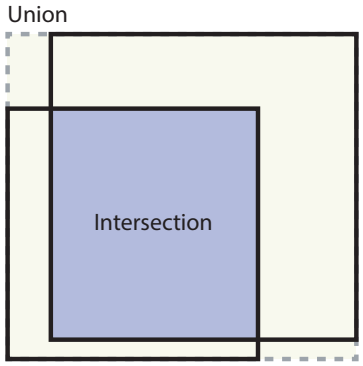


Figure 3.7: Two bounding boxes and their *intersection* and *union*.

Where $B_p \cap B_{gt}$ denotes the *intersection* of the predicted and ground truth bounding boxes and $B_p \cup B_{gt}$ their *union*. Both intersection and union are illustrated in Figure 3.7. The overlap threshold of 0.5 was set deliberately low by the authors of the PASCAL VOC dataset to account for inconsistencies and inaccuracies in the bounding boxes of the ground truth data. We use the same overlap thresholds since the argument of inaccuracies in ground truth data also applies to our work (see Section 3.1.3). Detections returned by a detection algorithm are assigned to ground truth objects satisfying the overlap criterion in order ranked by the (decreasing) confidence output. Multiple detections of the same object in an image are considered as false detection. For this work we use this exact definition of correct/incorrect detections when performing object detection in full images.

After the process of matching detections to ground truths we determine the number of correctly classified objects (true positive), detections that are incorrect (false positives) and ground truth objects that are not missed by the detector (false negatives). With addition of true negatives we can thus categorize each detection and ground truth box to one of the types summarized in Table 3.2. Based on the total number TPs, FPs and FNs a wide range evaluation statistics can be computed.

3.2.2 Precision-Recall Curves

Precision-recall curves are typically reported in machine learning research to determine the performance of classifier. The combination of precision and recall represent a trade-off: increasing one of these by modifying the detector threshold causes the other to decrease. So the two metrics are connected and together represent one dimension along which we can speak of the optimization of our system. This is very useful for setting the detection thresholds according to the desired output of the system. The definitions of *precision* and *recall* are given in terms of the total numbers of true positives, false positives and false negatives:

$$Precision = \frac{TP}{TP + FP} \qquad \text{(how many items are relevant?)} \tag{3.2}$$

$$Recall = \frac{TP}{TP + FN} \qquad \text{(how many items are detected?)} \tag{3.3}$$

In words the precision is the fraction of positive detections that is an actual positive hit (ground truth). The recall, also known as the hit rate, is the probability that a ground truth object is recognized by the detector. As apparent from these interpretations, the higher the better for detection accuracy. These definitions represent an evaluation measure independent of the detection algorithm and is therefore a common choice for comparing different machine learning algorithms.

4

Channel Features Detector

As we have seen in the discussion of related work, high-speed object detectors commonly use either a feature representation based on histograms of oriented gradients or Haar-features. HOG features are used frequently in top-performing object detectors [5], however the combination of HOG features in a SVM-classification framework is generally slower than cascade classification in a boosting framework. To put this in perspective, object detection using the combination of HOG+SVM is not particularly slow, but the specific normalization scheme and the high dimensional feature vectors hinder computational speedups. On the other hand, the detection framework building upon Haar features and boosting is among the fastest object detectors but lacks in detection quality due to the simpler feature representation.

More recently, a different type of visual features was proposed by Dollár *et al.* [16]. Both inspired by the work on histograms of oriented gradient descriptors [11] and Haar-features [88], *integral channel features* combine the computational efficiency of Haar features with the information richness of HOG features. These features have been shown to outperform HOG and Haar features in detection speed while maintaining similar detection rates. In the early stage of this research it was decided to use integral channel features as foundation of our detection framework. We motivate this decision by observing that – with a proper set of channels – the channel feature descriptors are closely related to HOG features and at the same time were reported as best performing with regard to detection speed [16, 55]. In this chapter we give a formal introduction to integral channel features as image descriptors in an object detection framework, and we discuss the AdaBoost learning framework. Together these two building blocks are the core components of our baseline detector which is introduced at the end of this chapter.

4.1 Integral Channel Features

Extracting robust features for object detection is the essential first step for building an object detector. There tends to be a trade-off between the richness of information features can capture and the computational efficiency at which they can be extracted. Rich feature

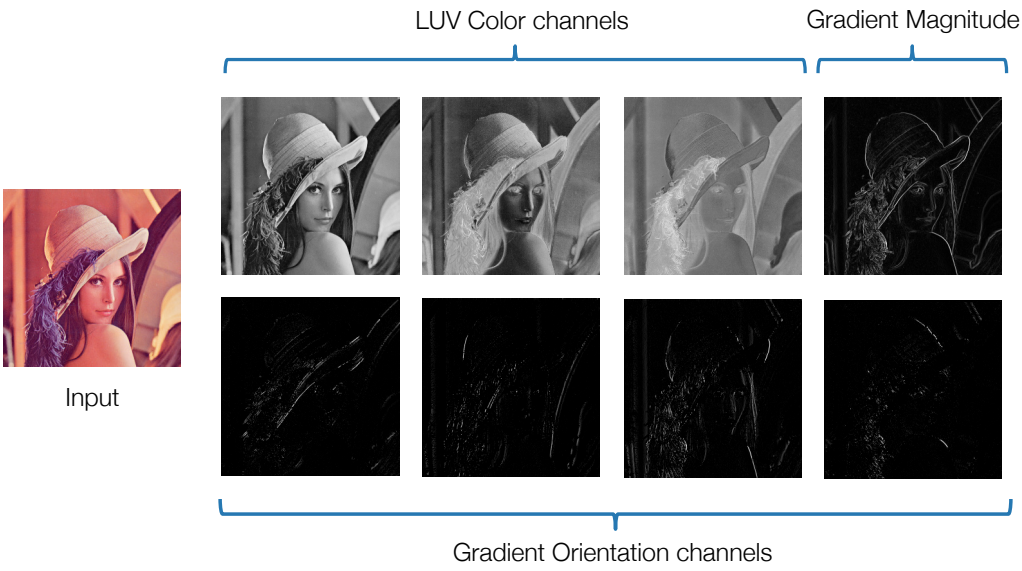


Figure 4.1: Multiple registered image channels $C = \Omega[I]$ computed for an example image. From these channels a rich feature set can be extracted, for example Haar features, approximations of HOG features, local sums and histograms.

representations can capture complex patterns in the data but typically require multiple normalization steps and a larger local-neighborhood search space. Generally speaking, more complex features require more computation power to be extracted from an image. Motivated by the simple and highly efficient feature extraction approach of Viola and Jones, Dollár *et al.* proposed the idea of using a larger variety of image channels for extracting a richer set of features while maintaining the computational efficiency. We will now give a general definition of channel features, present various channel types, discuss the use of integral images and thereafter we continue to work out the feature selection and learning process in the AdaBoost framework.

Using the concept of image channels, a broad family of features can be defined in a natural way. Let I be an input image and Ω any low-level spatial shift-invariant transformation. The application of transformation Ω on input image I creates a new filtered image or channel image, as its commonly referred as in object detection literature. We denote the channel image C obtained by transformation Ω as $C = \Omega[I]$. A transformation is shift-invariant if it produces the shifted output $g(x - a, y - b)$ when the shifted signal $f(x - a, y - b)$ is given as input to the system. In the context of images with pixel coordinates i, j the shift-invariance constraint is given as follows.

$$C(i, j) = \Omega[I](i, j) \Longleftrightarrow C(i + i_0, i + j_0) = \Omega[I](i + i_0, i + j_0). \tag{4.1}$$

The shift-invariance constraint is important in context of object detection using sliding-windows. Given I and I' related by translation, the corresponding channels $C = \Omega[I]$ and $C' = \Omega[I']$ are related by the same translation. As a result the transformation Ω only needs to be evaluated once on the entire image rather than separately for each overlapping detection window. In other words the transformation Ω is a registered map of the original

image, preserving overall image layout. Figure 4.1 displays eight different image channels extracted from an input image. Note that just like RGB images, the image channels can contain multiple layers indicated by the number k.

Given a set of image channels, a vast amount of image descriptors can be computed from the channels. For example pixel sums in rectangular regions, local histograms, and higher-order more complex combinations such as Haar-features. With this in mind, Dollár *et al.* [16] propose a general definition of **Integral Channel Features (ICF)**:

$$f_\Omega(I) = \sum_{i,j,k} w_{i,j,k}\, C(i,j,k) \tag{4.2}$$

where $i, j \in R$ denote pixels in a certain region and k indicates the channel layer. Using this general definition of image features, numerous local and global features can be expressed in this context. Examples include gradient histograms, linear filters, color statistics and others. Note that Haar-features can also be expressed using this definition by observing that VJ simply extract features from the intensity channel, and fixing the weights such that the adjacent rectangular windows are evaluated as difference.

This definition is very general and gives flexibility to assign weights to individual pixels and cover multiple channel layers. However in practice typically *first-order* channel features are used for object detection motivated by simplicity, speed and sufficient descriptiveness. For the application of pedestrian detection is was shown by experiments in [16] that using higher-order features has little positive effect on detection rate compared to first-order features. So for efficiency reasons, first-order features are preferred in the context of object detection. First-order channel features are simple pixel summations over fixed rectangular regions computed over a single layer. In other words, channel feature are simply become unweighted sums over a rectangular window $f_\Omega(I) = \sum_{i,j} C(i,j)$. Our baseline detector also uses first-order channel features as core component, we do not consider higher-order features in this work.

4.1.1 Channel Types

To build a strong feature representation, it is important to select an appropriate set of image channels from which the features f_Ω are extracted as rectangular pixel sums. Before focusing on channels used in our baseline detector, we briefly discuss a number of shift-invariant image transformations that are common in image processing and are candidates for building powerful image descriptors.

- ***Color Transformations.*** Given a RGB image as input, the simplest channel that can be computed is a linear combination of the color components, such as the grayscale version of the image. Additional color channels include conversion to the LUV color space which was designed to be perceptually uniform. This particular color space is popular for extraction of channel features since a change in value corresponds

roughly to the same perceptual difference over any part of the color space, *i.e.* color differences are represented more effectively.

- ***(Non) Linear Transformations.*** Linear filters are a simple and effective method for generating diverse channels. These include Difference of Gaussian (DoG) filters and Gabor filters, encoding texture and edge information respectively. Non-linear transformations are capable of extracting gradient magnitude channels and can operate as Canny filter for edge extraction.
- ***Histograms Transformations.*** Gradient histograms are a useful addition to gradient magnitudes since they capture the orientation of the gradient. Zhu *et al.* [94] propose the use of channel images for approximating HOG features. For each pixel of the input image the bin index is determined by the gradient orientation and the value summed in the bin is given by the gradient magnitude. In practice the number of orientation bins corresponds to an equal amount of "histogram bin images". Gradients at different scales can be computed by pre-smoothing the input image.

4.1.1.1 Preferred Channels for Object Detection

The important question is what channels are best suitable for extracting highly informative first-order channel features. In the original paper on integral channel feature [16] this question is answered for pedestrian detection. These results and the experiments performed in [58] demonstrate that the preferred set of channels, finding the right balance between information richness and computation efficiency, is given the following combination.

- LUV color channels (3×)
- Gradient orientation channels (6×)
- Gradient magnitude channel

Channel features are thus extracted from 10 image channels. Figure 4.1 displays eight of those channels – two gradient orientation channels are left to save space. The LUV color space is superior compared to other color spaces and the addition of more than six gradient orientation channels does not result in improvements of detection rate [16]. Our baseline detector also uses this set of channels. An overview of the gradient orientations used in our channel implementation is given in Figure 4.2. Note that "unsigned" gradients are computed, *i.e.* the direction of the gradient is not taken into account. To our best knowledge all detection frameworks building with channel features as object representation use this set of 10 channels. Intuitively using this set of channels makes sense since it represents all essential components with which we humans perceive objects [80]: color, orientations, texture and structure (gradient magnitude). Also it is appropriate to emphasize that the combination of gradient magnitude and gradient orientation channels together form an approximation of HOG features.

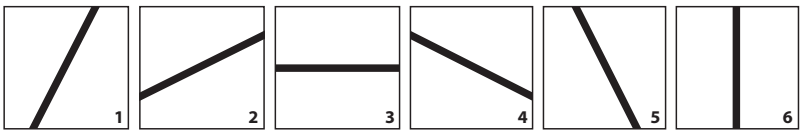


Figure 4.2: Edge orientations, corresponding to perpendicular gradient orientations indicating the centers of orientation bins used for channel computation. "Unsigned" versions of these orientations are used, *i.e.* the direction of gradient is not important. The orientations are numbered in agreement with our implementation.

4.1.2 Integral Images for Feature Extraction

As we will see in the next section, objects are represented by hundreds of channel features. For object detection we apply our trained model as sliding-window over the full image which requires feature extraction for each subwindow. Since our channel features f_Ω are computed as summations over pixel regions, a huge amount of pixel summations has to be performed. Although summations are a relatively cheap operation, the computation of a large amount of pixel sums is not feasible to run in real time. Fortunately the explicit summation of all pixels inside a rectangular window can be speed up by precomputing the *summed-area table* or integral image of a channel [9]. The use of integral images in the context of feature extraction was first introduced by Viola and Jones [88] for fast computation of Haar features. The definition of an integral image is very simple, each pixel $s(i, j)$ represents the sum of all pixels left and above the current pixel:

$$s(i, j) = \sum_{i'=0}^{i-1} \sum_{j'=0}^{j-1} f(i', j'). \tag{4.3}$$

Given the integral image, pixel summations over a rectangular window R spanning pixels $i_0, \ldots, i_1$ and $j_0, \ldots, j_1$ can be performed by four simple pixel lookups. The summation of all pixels inside a rectangular window is indicated in Figure 4.3 and is computed by the combining the following four pixel values from the integral image:

$$S(i_0, \ldots, i_1\,;\, j_0, \ldots, j_1) = \sum_{i \in R} \sum_{j \in R} f(i, j) = s(i_1, j_1) - s(i_1, j_0 - 1) - s(i_0 - 1, j_1) + s(i_0 - 1, j_0 - 1). \tag{4.4}$$

Thus after precomputation of the integral image, pixel summations can be performed extremely fast, requiring only four pixel lookups independent of the window size. The use of integral images for fast summing image windows also has a disadvantage: the memory cost of storing the integral images is larger than the original image. Since pixel summations over the entire image tend to become very large, 32-bit integer images are required for storing these huge numbers. Compared to regular grayscale images which only require 8-bits such images require a significant amount of memory which can be problematic for hardware with limited memory capacity. However, the speed increase offered by integral images typically outweighs this disadvantage.

3	2	7	2	3
1	5	1	3	4
5	1	**3**	5	1
4	3	2	1	6
2	4	1	4	8

3	5	12	14	17
4	*11*	**19**	24	31
9	**17**	28	38	46
13	24	37	48	62
15	30	44	59	81

3	5	12	*14*	17
4	11	19	24	31
9	17	28	38	46
13	24	37	**48**	62
15	30	44	59	81

Figure 4.3: Summed-area tables. **Left:** pixel values in the original image. **Center:** summed-area table. The red value in the table can be computed recursively from three adjacent values indicated in blue. **Right:** computation of area sum (green in original image) is done by combining the four values at the rectangle corners (purple) [83].

4.2 Learning Framework

The previous section focused on our feature representation of first-order channel features extracted from the three LUV color channels, six gradient orientation channels and the gradient magnitude channel. This feature representation offers highly informative object descriptors and can be extracted efficiently using integral images. In this section we shift our attention to the learning framework. We introduce the AdaBoost framework and explain how it can be used for feature selection and construction of a strong classifier. We emphasize our goal is to train the detector at a certain "model size" and use the trained model in sliding-window fashion to detect objects at all positions in an image. This is a typical instance of supervised learning: we are building a classification function from a set of labeled training instances and a feature representation.

4.2.1 Constructing a Random Feature Pool

Rather than carefully designing the features, a huge candidate feature pool is generated from which AdaBoost one-by-one selects the most discriminative feature in each training iteration. Feature mining is meant to overcome the effort and expertise necessary for careful feature design, and is shown to outperform manually designed features [17]. The first step in the training process is the generation of a candidate feature pool containing a huge amount of randomly selected features. First-order channel features are characterized by a random rectangular region (within the model size) and a channel index in the range 0-9 corresponding to one of the 10 image channels. In our implementation the rectangle location, dimensions and channel index are sampled uniformly at random. The only requirement for the randomly generated features is the minimum size; the rectangular region should contain at least 25 pixels in order to guarantee its usefulness. Smaller features are unlikely to contain useful information and therefore only slow down the training process.

For our application of rear-view car detection, we train our model at a fixed size, for example 64×64 or 80×80 pixels. The number of channel features that can be extracted for such model sizes is virtually endless. To guarantee that the learning algorithms can

select a strong set of features for describing the objects, the initial feature pool should be large enough. However training time increases linearly with the number of features in the candidate feature pool. Experiments by Dollár *et al.* [16] report that 30 000 candidate features is sufficient for training a strong pedestrian detector. In Chapter 7 we perform experiments to determine the sufficient number of features in the candidate feature pool for our application.

After random generation of the candidate feature pool we employ Discrete AdaBoost [31] for both feature selection and construction the strong classifier for classification of image subwindows. These two components are elegantly combined in the boosting framework, very similar to the learning process of Viola and Jones [88]. Next, we continue our discussion by describing how AdaBoost is utilized for feature selection and training a classifier. The terms introduced in the next section are also important for understanding Chapter 5 which builds upon the AdaBoost framework.

4.2.2 Discrete AdaBoost

The popular *boosting* framework in machine learning describes a method of combining the performance of many "weak" classifiers into a powerful "committee" to form a strong classifier. Boosting was proposed in the computational learning theory literature, originally by Kearns [43] and Schapire *et al.* [31]. Boosting is motivated by the question whether a set of weak classifiers can create one strong classifier. A weak classifier is just slightly better than random guessing, whereas the output of the strong classifier should be highly correlated with the true label. While boosting has evolved over the years, we describe and implement the Discrete AdaBoost version for two-class classification of subwindows in images. Not all details in this section are important for our classification framework, however we do want to give the reader a clear understanding of why boosting works.

Let our training data be denoted by $(x_1, y_1), \ldots (x_N, y_N)$ with x_i a vector valued feature and the $y_i \in \{-1, +1\}$ the corresponding label of the object. The AdaBoost procedure trains weak classifiers $h_t(x)$ on the weighted versions of the training samples, giving higher weight to examples that are currently misclassified [32]. This is done for a number of training iterations T, and then the final classifier is defined as a linear combination of the weak classifiers from each stage. The final classification score is denoted by H_T and defined as:

$$H_T(x) = \sum_{t=0}^{T} \alpha_t \cdot h_t(x). \tag{4.5}$$

Where each $h_t(x)$ is a weak classifier producing values -1 or $+1$ and α_t are constants related to the strength of the weak classifier of that stage. The corresponding prediction on the label of an object x is sign $(H_T(x))$. Note that for Discrete AdaBoost each weak classifier produces the label $\{-1, +1\}$ rather than a numerical value indicating the "confidence" in the prediction as is the case with Real AdaBoost. The detailed description of Discrete AdaBoost is given in Algorithm 1.

Algorithm 1 Discrete AdaBoost [31]

1: Let the training data be denoted by $(x_1, y_1), \ldots, (x_N, y_N)$.
2: Start with weights $w_i = 1/N$ for $i = 1, \ldots, N$
3: **for** training stages $t = 1, 2, \ldots T$ **do**
4: Fit the weak classifier $h_t(x) \in \{-1, +1\}$ using weights w_i on the training data.
5: Compute the weighted error $\epsilon_t = \sum e_i \cdot w_i$ over all samples and where $e_i = 0$ if the sample is classified correctly by $h_t(x)$ and $e_i = 1$ otherwise.
6: Compute the coefficient $\alpha_t = \log((1 - \epsilon_t)/\epsilon_t)$.
7: Update the weights, set $w_i \leftarrow w_i \cdot \exp(\alpha_t \cdot e_i)$ for $i = 0, \ldots, N$.
8: Renormalize the weights so that $\sum_i w_i = 1$.
9: **end for**
10: Output the classifier sign $\left(\sum_{t=1}^{T} \alpha_t \, h_t(x)\right)$.

As we see from Algorithm 1, the training procedure increases the weights of the observations misclassified by $h_t(x)$ by a factor α_t that depends on the weighted training error. It was shown by Friedman *et al.* [32] that AdaBoost algorithms can be interpreted as stage-wise estimation procedure for fitting an additive logistic regression model. Logistic regression is a popular approach used in statistics, and for a two-class problem has the form:

$$\log \frac{P(y = +1 \mid x)}{P(y = -1 \mid x)} = \sum_{t=1}^{T} h_t(x). \tag{4.6}$$

In AdaBoost training the exponential loss criterion $\mathcal{L}(H_T)$ is minimized, motivated as an upper bound on the misclassification error [73]:

$$\mathcal{L}(H_T) = \sum_i e^{-y_i H_T(x_i)} = \sum_i e^{-y_i \sum_{t=1}^{T} h^{(t)}(x_i)}. \tag{4.7}$$

It can be shown [32] that the exponential loss $\mathcal{L}(H_T) = \sum_i e^{-y_i H_T(x_i)}$ is minimized at

$$H_T(x) = \frac{1}{2} \log \frac{P(y = +1 \mid x)}{P(y = -1 \mid x)}. \tag{4.8}$$

Which corresponds to optimizing the logistic regression given in (4.6). Therefore the Discrete Adaboost algorithm builds an additive logistic regression model for minimizing the exponential loss function $\mathcal{L}(H_T)$. In other words, at each stage the weak classifier h_t is selected according to minimizing the exponential loss function:

$$h_{T+1} = \arg\min_{h_t} \mathcal{L}(H_T + h_t). \tag{4.9}$$

Notice that after each update to the weights, the weighted misclassification error of the most recent weak learner is at most 50%. Schapire *et al.* [73] give the interpretation that the weights are updated to make the new weighted problem maximally difficult for the next weak learner. Interested readers are referred to Friedman *et al.* [32] who present an excellent discussion on the statistical view of boosting. Some of the results described in this section are also important for Chapter 5 in which an extension to AdaBoost is

described for providing classification speed-up. After discussing the general idea behind boosting we now turn our attention to the use of Discrete AdaBoost for feature selection and training a strong classifier.

4.2.3 AdaBoost for Feature Selection

Viola and Jones prove the effectiveness of using the Discrete AdaBoost algorithm to select the *single* rectangular feature which best separates the positive and negative training examples at each stage. This method was also adopted by Dollár *et al.* [18] for selecting the best channel features from the candidate pool. Our learning framework is similar. In the following discussion we make a change of notation, where for the discussion in the previous sections object labels where defined at $y_i = \{-1, +1\}$ we make a change to $y_i = \{0, 1\}$ to follow the notation in the original paper by Viola and Jones [88]. An detailed overview of the Viola-Jones feature selection and training procedure is given in Algorithm 2.

So far we have not specified the weak learner $h_t(x)$, in practice this is usually a simple decision tree. For our work we use simple depth-1 decision trees, better known as *decision stumps*. Stumps are single-split trees with only two terminal nodes. In the context of our work a decision stump is defined by a single channel feature f_j, an threshold τ and a parity p_j indicating the direction of the inequality sign [88]. The formal definition of our weak learner thus becomes:

$$h_j(x) = \begin{cases} 1 & \text{if } p_j f_j(x) < p_j \tau_j \\ 0 & \text{otherwise} \end{cases}. \tag{4.10}$$

In each training round the best weak learner $h_j(x)$ is determined for all candidate features. Although we have defined h_j as a decision stump, the weak learner can also be more complex involving more features, *i.e.* higher-order decision trees. Once the best possible weak learner for each feature j is determined, AdaBoost selects the weak classifier with the lowest weighted error over all training samples. The selected weak classifier is denoted by h_t with associated error ϵ_t. In other words, at each stage a single features is selected which minimizes the exponential loss function over the weighted training examples. After determining the best weak classifier for all features, the error of all weak classifiers is computed with respect to the weights from the previous learning round following the AdaBoost algorithm:

$$\epsilon_t = \sum_{i=1}^{N} w_i |h_t(x_i) - y_i|. \tag{4.11}$$

Where N is the total number of samples in the training set and w_i indicates the weight of a sample at the current training iteration. Although the notation is somewhat different, this directly corresponds to the error given in Algorithm 1. At this stage we have computed the lowest possible error rate for each of the decision stumps associated with the candidate features. The AdaBoost algorithm now greedily selects the decision stump h_t with the

best error rate ϵ_t. The training iteration concludes by updating the weights of all samples such that the weight of samples classified correctly by weak learner h_t is decreased.

The algorithm terminates when a predefined number of T weak classifiers is trained or alternatively when the classification error on the training sets drops below a certain threshold. After training is finished and the T most informative features are selected, the strong classifier is constructed such that the weight assigned to each weak classifier is inversely proportional to the training error ϵ_t of that stage. The final strong classifier is thus constructed using T hypotheses each using only a single feature, this is reflected in Equation (4.15) occurring in the AdaBoost algorithm.

Algorithm 2 Viola-Jones feature selection and training [88]

1: Let the training data be denoted by $(x_1, y_1), \ldots, (x_N, y_N)$ with labels $y_i \in \{0, 1\}$

2: Start with weights $w_1 = \frac{1}{2m}$ and $w_1 = \frac{1}{2l}$, where m and l indicate the number of positive and negative samples respectively.

3: **for** training stages $t = 1, 2, \ldots T$ **do**

4: For each feature j, train a weak classifier h_j which is restricted to using this single feature (details in text). The error of the weak classifier is a weighted sum of misclassified samples:

$$\epsilon_j = \sum_i^N w_i \, |h_j(x_i) - y_i| \tag{4.12}$$

5: Choose the classifier h_t, which corresponds to the lowest error ϵ_t.

6: Update sample weights by decreasing the weight of correctly classified samples:

$$w_i \leftarrow w_i \cdot \beta^{1-e_i} \tag{4.13}$$

Where $e_i = 0$ if sample x_i is classified correctly and $e_i = 1$ otherwise.
The factor β_t increases when the error of the weak classifier is larger.

$$\beta_t = \frac{\epsilon_t}{1 - \epsilon_t} \tag{4.14}$$

7: Renormalize the weights so that $\sum_i w_i = 1$.

8: Save decision stump h_t and corresponding values ϵ_t, β_t.

9: **end for**

10: Output the **strong classifier** is given by

$$H_T(x) = \begin{cases} 1 & \text{if } \sum_{t=1}^{T} \alpha_t \, h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases} \tag{4.15}$$

Where factor α_t assigns more weight to better weak classifiers, since $\alpha_t = \log \frac{1}{\beta_t}$.

Figure 4.4: Typical values of ϵ, α and β during the boosting training process. This figure illustrates that for early stages AdaBoost selects the most discriminant features and that these stages receive most weight α_t in the final strong classifier. The spiking appearance in later stages can be explained by the reweighting of training samples after each stage.

4.2.4 Learning Discussion

As we have seen in the last two subsections, during the learning process a crucial role is played by the lowest error ϵ_t for each stage. From the error rate, the weight factor β_t is computed which is then used for assigning importance α_t to the corresponding weak classifier of that stage.

$$\alpha_t = \log \frac{1}{\beta_t} = \log \frac{1 - \epsilon_t}{\epsilon_t}. \tag{4.16}$$

In words, this means that the weight assigned to each weak learner in the strong classifier is inversely proportional to the error of that stage. Better weak classifiers receive more importance in the final classifier. In Figure 4.4 we plot the characteristic for an example training procedure using AdaBoost and channel features. From this figure we observe that the weak classifiers in the early stages receive most weight in the final classifier. This is not surprising since the best features are selected first. The spiky appearance of the values is due to the sample weight update after each training stage.

Using the discussed boosting approach, the majority of features is discarded very aggressively, while maintaining only a small set of features that is best suited for the classification problem. An additional advantage of this feature selection process is that the features selected in early stages of the learning process are easily interpretable. For example when selecting the most discriminating Haar features using boosting we observe that the best features focus on the area around the eyes and nose [88].

AdaBoost offers several advantages in comparison to other classifiers: it is simple, its implementation is straightforward and the time complexity of each boosting round is

rather low. When using decision stumps the time complexity of each boosting round is $O(mN)$, where m and N represent the size of the feature space and N the number of training examples [57]. Of course training can still become slow when the feature space is very large. Boosting also has some disadvantages compared to other machine learning methodologies. The main drawbacks of the algorithm are the need to select the number of boosting rounds T, choosing of the weak classifier and the performance in the presence of noise. The choice of the number of boosting rounds T and the type of the weak classifier are crucial to good performance of the algorithm. Larger values of T can lead to overfitting whereas an insufficient number of boosting rounds typically results in high false positive rates.

Probably the most serious drawback of boosting is its poor performance in the presence of noise in the dataset. It has been shown empirically that noise can seriously damage the classification accuracy [52]. The distribution weights assigned to examples that are harder to classify substantially increases with the number of boosting rounds by nature of the algorithm. However this causes harder samples to end up dominating the selection of following weak classifiers since they receive most of the weight. Several solutions have been proposed to address these issues, for example by using a "less aggressive" loss function compared $\mathcal{L}$ to the exponential loss function of AdaBoost to penalize less incorrectly classified points at each boosting iteration [57]. For training on datasets with significant level of noise one can consider the use of regularized boosting [54].

4.3 Classification of Subwindows

The result of a training procedure is a strong classifier $H_T(x)$ given in Equation (4.5). This classifier "contains" T channel features that are selected as best descriptors for the object detection task. Given the linear combination of weak classifiers, the actual classification process is straightforward. The training detector of a given model size is employed in sliding-window fashion. For each location the T channel features are extracted using the precomputed integral images. The classification process starts with initialization of sample *trace* $H_t = 0$. Features x selected by the boosting algorithm are evaluated one at a time. Starting with $t = 0$ for increasing stages, the decision stumps $h_t(x)$ given in (4.10) predict a label of the sample. If the weak learner predicts the class label +1 the sample trace H_t is increased by factor α_t corresponding to the current weak learner. Therefore the sample trace is a non-decreasing curve as function of the stages t. At the final stage of the sequence of weak classifiers, the final label $y \in \{0, 1\}$ is predicted by thresholding the score H_t:

$$H_T(x) = \begin{cases} 1 & \text{if } \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases} . \tag{4.17}$$

To illustrate the classification process we plot some example traces in Figure 4.5. This classification process is simple but requires an excessive number of features to be extracted

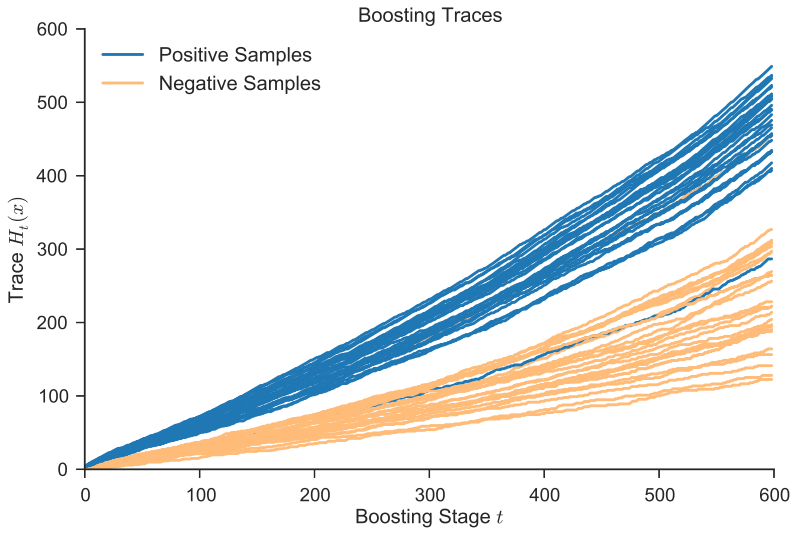


Figure 4.5: AdaBoost traces $H_t(x)$ on our TomTom test dataset consisting of car and background samples. We randomly selected 30 traces of both classes for display purposes. Note how the gap between positive and negative samples increases by progressing through the stages.

and evaluated. In Chapter 5 we discuss the implementation of methods for speeding up the classification process.

4.4 Baseline Detector

So far in this chapter we have discussed our feature representation and the boosting framework for feature selection and classification of subwindows. These components form the foundation of our baseline detector, which we introduce in the final section of this chapter. We also take opportunity to give some details of our implementation, and present some preliminary speed-benchmark results. The complete architecture of our baseline detector is displayed in Figure 4.6.

Detection of objects of interest in a full image starts with computing the image pyramid and extracting the image channels for each scale, this is also referred to as the *feature pyramid*. The feature pyramid is a multiscale representation of an image I, where channels $C_s = \Omega[I_s]$ are computed for a large number of scales s. Scales are typically sampled evenly in log-scale, starting at $s = 1$, with typically 4 to 8 scales per octave [13]. Computation of the image pyramid and image channels is performed by our fast GPU implementation. This standard approach for multiscale detections in a framework using channel images is illustrated in Figure 4.7.

After computing the channels, the results are sent back to the CPU which then starts the classification of all subwindows over the complete image pyramid using the sliding-window. The features required for classification are extracted from the precomputed

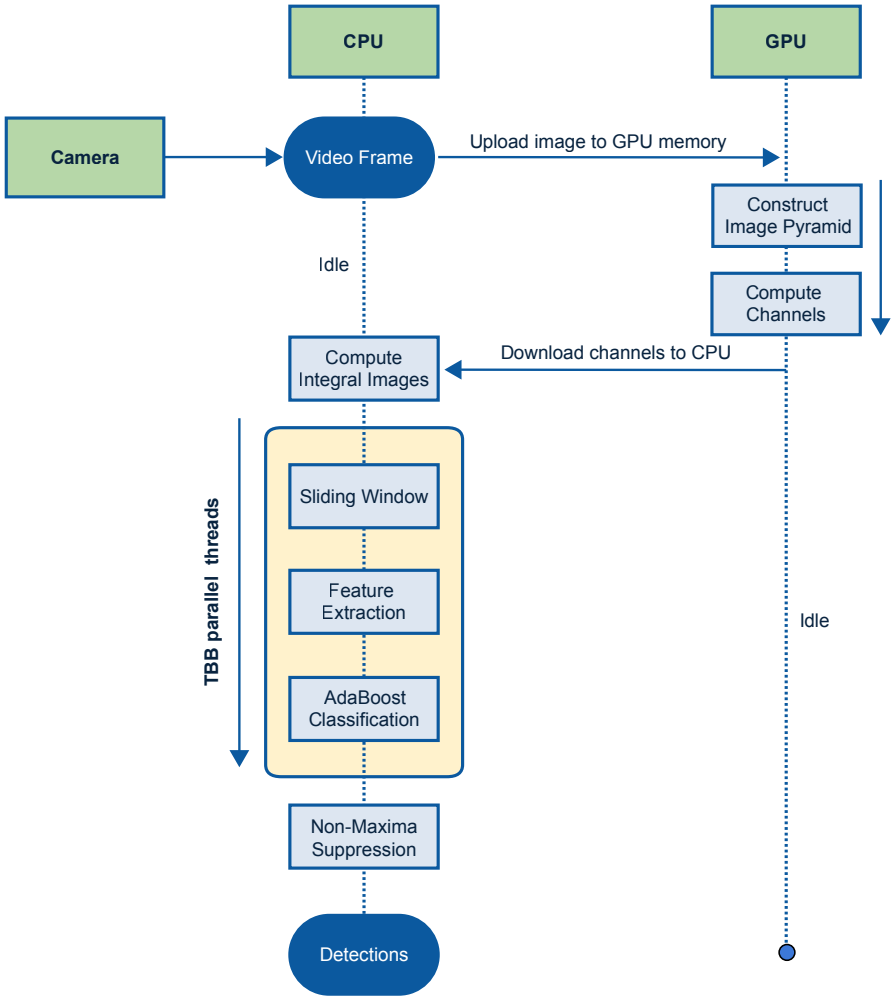


Figure 4.6: The detection pipeline of our baseline detector. In our current implementation the GPU is only used for constructing the image pyramid and applying filter kernels to extract the 10 image channels. Feature extraction, multiscale approximations and classification are performed on the CPU in a parallel implementation using TBB.

integral images, hence the feature extraction is very efficient. Subwindows that are classified as object, are saved as detection candidates. Once the entire image is processed, nearby candidate detection windows are merged together by applying non-maxima suppression (NMS). Our implementation uses a simple NMS algorithm of which the details are given in [16].

4.4.1 Implementation Details

We conclude this chapter with some brief notes on our implementation of the baseline channel features detector. The code base for our detector is written in C++11 building upon the OpenCV framework for computer vision. Only basic functionality of OpenCV is used, *e.g.* matrix operations, simple filtering and image I/O. Upon receiving an image from camera or disk the detector uploads the images to the graphical processing unit

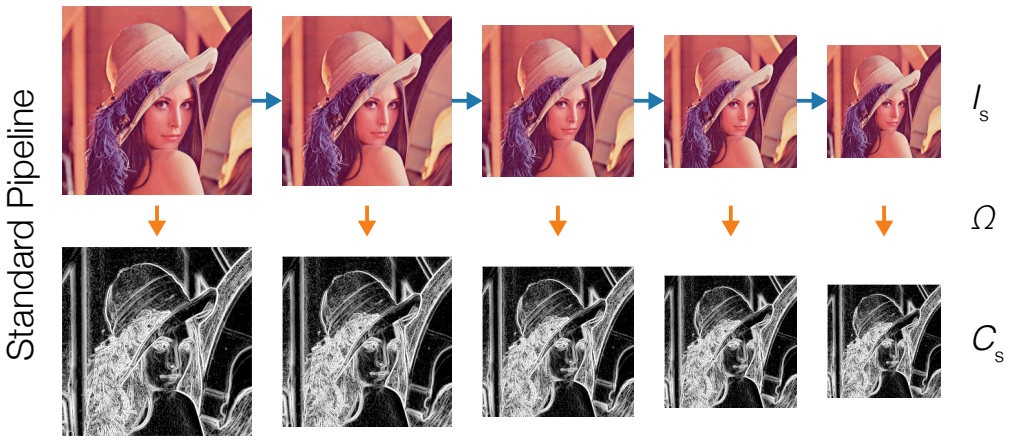


Figure 4.7: Standard multiscale detection pipeline. Image channels C_s are computed for every scale s by computing the image pyramid and extracting the channels for each scale. The pyramid of image channels is also referred to as the feature pyramid.

(GPU). The GPU computes the integral channels of the image using fast OpenCL kernels, the channels are then downloaded back to CPU memory. Classification is performed on the CPU but is parallelized where possible.

Input images are represented in RGB color space and no pre-processing are included, *i.e.* no gamma or color normalization is applied. Detector performance is sensitive to the method with which gradients are computed. Typically image gradients are computed by smoothing the image with a Gaussian kernel and sliding over a discrete derivative mask. Several derivative masks are available such as Prewitt, Sobel and Scharr operators [64]. In the original work by Dalal and Triggs [11] on HOG descriptors the authors perform an extensive study to determine the optimal smoothing parameter σ and derivative kernel. Of all combinations tested, simple 1D masks $[-1, 0, +1]$ and $[-1, 0, +1]^T$ without smoothing give best performance in context of pedestrian detection. Using larger masks, a significant decrease in detection rate is reported and smoothing also has negative influence on the results. Based on these results we implement the same 1D kernels for gradient computation in both directions and we apply no pre-smoothing. For color images it is common to apply the gradient kernels to each color channel and select the one with the largest response. Since we have already computed the grayscale image as part of our LUV channels (L represents luminance) in our implementation we perform convolution of this channel with our gradient kernels.

Our implementation also features various tricks for improving computational efficiency. Small optimizations at this level do matter when color conversions and gradient orientation computations are performed for each pixel in every video frame. For example our implementation features additional speed-ups by optimizing cube root $\sqrt[3]{x}$ calculations in LUV color conversion and improving gradient orientation calculation. The cube root computation can be speed-up by including Halley's method [33] for fast approximation of the root which is significantly faster than the exact computation with small error

compared to the exact result. Gradient orientations are typically computed by evaluating $\Theta = \text{atan2}(G_y, G_x)$ where G_y and G_x denote the outputs of the gradient masks in both directions. Several methods have been suggested for speeding up the arctangent computation [66]. In our implementation we have included the fixed point arctangent computation with self normalization originating from digital signal processing [77]. The extraction of all 10 channels using our GPU implementation and downloading the channels back to CPU memory runs at 40 fps for 1024×768 images and 92 fps for 640×480 images our hardware. Note that this is almost five times as fast as the original implementation [16] reporting channel extraction running at 40 fps on 320×240 images. At this stage, benchmarks (Figure 4.8) show that the most time is spent on feature extraction and classification of subwindows. Therefore in the next section we focus on reducing classification and feature extraction time.

4.5 Chapter Summary

In this chapter we have introduced our baseline detector building upon integral channel features and the boosting framework. The combination of diverse, informative channels and the fast feature computation using integral images make integral channel features a suitable candidate for high-speed object detection. The AdaBoost learning framework serves two purposes in our baseline classifier: feature selection from an huge pool of candidate channel features and constructing a strong classifier for classification of subwindows. The combination of ICF and AdaBoost opens up the door to high-speed object detection, although the baseline detector itself is not yet capable of running in real time. In the following two chapters we suggest the implementation of approximation techniques for significantly increasing the detection speed.

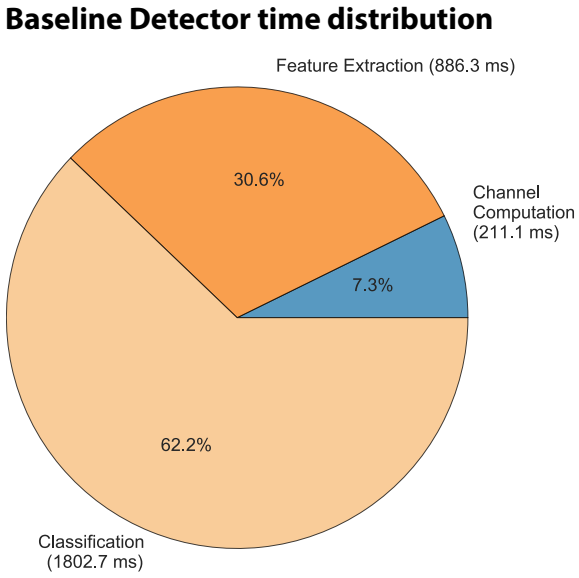


Figure 4.8: Per-component time contribution baseline detector.

5

Sequential Decision Processes

The experiment that conclude the previous chapter shows that the time spent on actual classification of all windows in an image consumes a significant amount of the total processing time. Computing feature values is extremely efficient using integral images, and the cost of evaluating a weak classifier can also be done with little computational resources. However since the strong classifier can have up to 2000 weak classifiers, and the number of image windows over all scales is extremely large, the total time spent of feature extraction and classification is a significant contribution. In this chapter we describe a sequential decision making process for two-class classification with the goal of speeding up classification of subwindows in an image.

Viola and Jones [88] already pointed out the need for decreasing classification time to achieve practical high-speed object detection. By introducing their so-called *attentional cascade* as extension of the AdaBoost framework the object detector is capable of rapid rejection of non-promising negative subwindows. The key insight is that smaller, and therefore more efficient, strong boosted classifiers can be constructed for rapidly rejecting a large fraction of negative subwindows in an image. Classifiers at later stages in the cascade are more complex and focus on achieving high overall precision. By proceeding through the cascade the subsequent classifiers become more difficult to pass. A positive result from a classifier triggers the following classifier, and so on. When a subwindow is assigned a negative label, the window is immediately rejected and excluded from further evaluation. This structure reflects the fact that the majority of windows is negative, as it attempts to reject as many negatives as possible at the earliest stage. The VJ cascade design is illustrated in Figure 5.1. The original cascade design has the disadvantage that the information obtained from evaluating previous stages is discarded as it passes to the next stage. Consequently, the decision to accept or reject an instance at a given stage does not take into account how well the instance performed in prior stages [7].

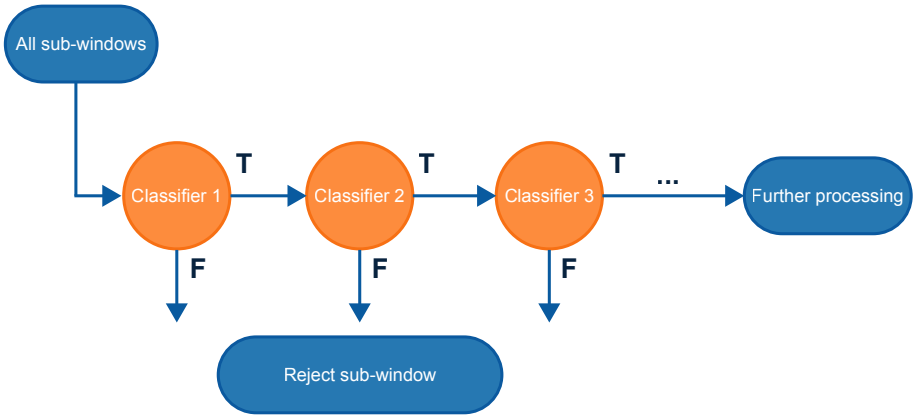


Figure 5.1: Original Viola-Jones cascade design [88]. Classifiers become more complex at later stages. Early stages focus on rejection of non promising subwindows.

In the formal definition of the VJ framework, the boosting cascade consists of K successive classifiers each with a false positive rate f_i and detection rate d_i. The false positive rate and detection rate of the aggregated classifier is then given by $F = \prod_{i=1}^{K} f_i$ and $D = \prod_{i=1}^{K} d_i$ respectively. Consequently, for each layer of their boosting cascade, the classifier is adjusted to a very high recall to make sure no relevant objects are lost during the classification process. For example, to obtain a 95% detection rate for the final classifier consisting of 10 smaller classifiers, each classifier should have a recall rate of on average $\sqrt[10]{0.95} = 0.9949$. However such high recall rates come at the price of sharp decreases in precision. This trade-off poses the difficulty of per-stage parameter tuning which tends to be difficult. It was shown in [91] that the computational cost involved with evaluating a classifier in the cascade is a function of the false positive rate f_i and the number of weak classifiers T for this classifier. For a given detection task with specified requirements on F and D the problem concerns finding the optimized set of f_i and T-values. Since this is a non-trivial optimization task, methods referred to as soft-cascades address this problem.

5.1 Introduction to Soft-Cascades

The structure of soft-cascades is different since only one classifier of T weak learners is utilized in stage wise fashion. By determining stage rejection thresholds that control the early rejection of samples, the same target false positive rates and detection rates can be achieved. The foremost advantage of this approach is that the difficult optimization process of parameters f_i and T per classifier becomes unnecessary. The basic approach consists of two main ideas: (1) to generalize each stage from a binary-valued decision process to a scalar-valued decision process, *i.e.* the decision function is proportional to how well the given instance passes the stage and the relative importance of the stage; and (2) the decision function takes into account all preceding stages rather than only the most recent as with the original VJ cascade [7].

Most notable publications regarding such cascades are the closely related works by Bourdev *et al.* [7], Šochman *et al.* [79] and Zhang *et al.* [93]. While these methods all use different terminology, we indicate them as *soft-cascades*, a term introduced in [7]. Each of these works focus on learning stage rejection thresholds $\theta(t), t \in \{1, \ldots, T-1\}$ for the final boosted cascade. The score or *sample trace* $H_t(x)$ of a detection window x is the partial sum up to the current stage t of the strong classifier, as given by the AdaBoost algorithm in Equation (4.15):

$$H_t(x) = \sum_{t' \leq t} \alpha_{t'} \cdot h_{t'}(x) \tag{5.1}$$

The trace scores are non-decreasing; a number of example traces is displayed in Figure 4.5 of the last chapter. The stage rejection thresholds $\theta(t)$ that these papers attempt to determine are critical to the performance and speed of the complete classifier. If during classification the cascade score drops below the stage threshold, the subwindow is immediately pruned, *i.e.* the window is rejected when $H_t(x) < \theta(t)$. This generalizes each boosting stage classification to be scalar-valued, rather than binary-valued. The decision function is proportional to how well the given instance passes the previous stages weighted with the corresponding importance of that stage. Incorrect thresholds can result in a significant amount of false negatives when positive samples are incorrectly rejected. On the other hand, as we will see later, the stage rejection thresholds are crucial in achieving real-time performance for object detection. In the remainder of this section we briefly discuss some of the strategies proposed for determining stage rejection thresholds.

Rejection Distribution Vectors. Bourdev *et al.* (2005) [7] introduce the concept of *rejection distribution vectors*. The vector stores the minimum fraction of positive samples v_t that can be missed at each stage, *i.e.* $\mathbf{v} = (v_1, \ldots, v_T)$. The sum of its elements is $1 - D$ where D is the target detection rate of the classifier. The distribution of the vector elements controls the trade-off between expected evaluation time and false positive rate. The authors propose an algorithm that returns the execution time and false positive rate for a particular rejection distribution vector. Using this algorithm they continue to empirically determine the vector $\mathbf{v}$ that minimizes the FPR and target detection rate given the target detection rate D. Based on their experiments they conclude that the specific shape of the distribution vector is not important but rather the distribution of its elements towards the beginning and towards the end of the vector. Elements for early stages corresponded to faster classifiers, while elements for later stages are in relationship with lower false positive rates [7]. Based on these findings the authors suggest using an exponential distribution $v_t \propto e^{\pm t}$ which favors speeds over accuracy at early stages while favoring accuracy over speed for later stages. However the family of exponentially distributed rejection vector is arbitrary and certainly not optimal [93].

Multiple Instance Pruning. In a different approach by Zhang *et al.* (2008) [93] the authors formulate the object detection question as *multiple instance learning* problem and propose the multiple instance pruning (MIP) soft-cascade algorithm. In their algorithm positive detection windows are collected inside bags and the learning framework is given the

freedom to select at least one example of the true positive examples. In a post-training stage after calculating the strong AdaBoost classifier, the rejection thresholds are determined so that every positive sample detected by the original strong classifier is guaranteed to be detected by the soft-cascade. This is realized by running the initial detector over all windows that match with ground truth objects and putting all windows that belong to the same ground truth in a bag. The MIP process is fully automatic and requires no assumptions of probability distributions, statistic independence or intermediate rejection target rates.

WaldBoost. The methods discussed above set the rejection thresholds empirically and cannot proven to be optimal. Šochman *et al.* (2005) [79] approach the problem fundamentally different by formulating the classification problem as sequential decision process that is characterized by the error rate and average evaluation time of the classifier. After formulating the problem this way, the authors show that the optimal strategy, in terms of the shortest average decision time subject to requirements on the error rates (FP and FN rates), is given by Wald's sequential probability ratio test. To our best knowledge this is the only work that explicitly addresses the optimal trade-off between requirements on the error rates and average classification time. In contrast to the related works this method of setting the threshold retains statistical validity. The disadvantage of WaldBoost is that setting the thresholds relies on a process involving probability density estimation. This poses an empirical risk since the process of density estimation is certainly not trivial in practice.

5.1.1 Method of Choice and Motivation

From the variety of classification models discussed above there are two methods that stand out: the MIP-based detector by Zhang *et al.* [93] and the WaldBoost algorithm proposed by Šochman *et al.* [79]. Detectors such as Viola-Jones, the Boosting Chain by Xiao *et al.* [91] and the soft-cascade of Bourdev *et al.* [7] all require manual tuning of the parameters. The methods that stand out do not require manual parameter tuning, since only specification of the final requirements on the error rates are required as input parameter. Zhang *et al.* [93] formulate the process of learning stage thresholds as multiple instance learning problem (MIP). This approach is successfully adopted in detection frameworks [16, 13] due to its simplicity and effectiveness. For our work we choose to adopt **WaldBoost** as soft-cascade approach to speedup classification. Experiments [93] have demonstrated that WaldBoost is faster than MIP for rejecting subwindows, and we appreciate the statistical foundation of WaldBoost, *i.e.* the sequential probability ratio test used for calculating stage thresholds is proven optimal, while the other methods are more designed based on intuition and empirical observations.

In their work, Zhang *et al.* [93] report a performance comparison in terms of detection rate and false positive rate between the discussed soft-cascade approaches. Comparing the detection rates of MIP and WaldBoost shows a difference of about 2% in favor of MIP. However we emphasize that the authors themselves are authors of the MIP algorithm,

hence the evaluation might not be completely unbiased. In an additional comparison the authors show that the average number of features visited per window (face detection) is the lowest for the WaldBoost detector compared to all other soft-cascade approaches. The detection rates are lower compared to MIP, however since we are more interested in achieving high speed than the highest detection rates which motivates our choice of using WaldBoost. In the following sections we discuss the ideas behind WaldBoost starting from the definition of sequential decision processes working towards the use in an object detection framework.

5.2 Two-Class Sequential Decision Processes

In this section we formulate the two-class classification problem as a sequential decision process. This formal definition is the foundation of WaldBoost for speeding up the classification process in context of boosting and object detection. Let x be an object characterized by its hidden class label $y = \{-1, +1\}$ that we want to predict using an ordered sequence of measurements $\{x_1, x_2, \ldots, x_t\}$. A *sequential decision strategy* S is a sequence of decision functions $S = \{S_1, S_2, \ldots\}$ for which at any stage one of the following three decisions is made based on the t measurements available so far:

$$S_t : (x_1, x_2, \ldots, x_t) \rightarrow \{-1, +1, \sharp\} \tag{5.2}$$

Where $\sharp$ indicates 'continue' when the object is undecided from the first t measurements. The decision function indicates that there are three possibilities in each stage of the decision process: label the object as positive, label the object as negative, or continue by making an additional observation. In a stage-wise fashion the statistical experiment continues until either the first or second decision is made. An important feature of the sequential test is that the number of observations required is not predetermined; in fact the number of observations itself is a random variable since at any stage the decision of terminating the the process depends on the results of observations previously made. The decision time of an object is defined as the smallest t for which a decision is made:

$$T_s(x) = \arg\min_t \left(S_t \neq \sharp\right) \tag{5.3}$$

The result of such decision will be denoted by $S(x) \in \{-1, +1\}$. A sequential strategy S is characterized by its error rates α_s and β_s and its average evaluation time $\overline{T}_S$ given by

$$\overline{T}_S = E\left[T_S(x)\right] \tag{5.4}$$

where the expectation or time-to-decision is determined over $T_s(x)$ weighted with the probability of occurrence for object x. The two error rates corresponding to making an incorrect decision during the process are expressed in terms of false negative rate α_s and false positive rate β_s. In the context of object detection and the sequential strategy, the error rates can be expressed as the probability of making an incorrect decision:

$$\alpha_s = P\left(S(x) = -1 \mid y = +1\right) \tag{5.5}$$

$$\beta_s = P\left(S(x) = +1 \mid y = -1\right) \tag{5.6}$$

Given user-specified requirements on these two error rates α and β, the sequential decision process can be expressed as optimization problem. The formal decision of the optimal sequential decision making process for requirements α and β is defined by [79]:

$$\begin{aligned} S^{\dagger} &= \arg\min_{S} \overline{T}_S \\ \text{s.t.} \quad & \beta_S \leq \beta, \\ & \alpha_S \leq \alpha. \end{aligned} \tag{5.7}$$

The question that remains is how we find the optimal solution for this strategy. Fortunately a solution to this problem was derived by Abraham Wald [90]. In this work an optimal solution to this problem is found in terms of the sequential probability ratio test. In the next section we will give an overview of the work by Wald and describe the optimal solution to the optimization problem (5.7).

5.3 Sequential Probability Ratio Test

Following the work by Wald [90] on sequential decision processes and corresponding optimal strategies, statistically speaking we are testing our *null hypothesis* H_0 against the *alternative hypothesis* H_1 given by:

$$\begin{aligned} \text{null hypothesis } H_0 &: \text{object belongs to positive class } +1 \\ \text{alternative hypothesis } H_1 &: \text{object belongs to negative class } -1 \end{aligned}$$

The objective is to formulate a decision strategy such that the probability of accepting H_1 given that H_0 is actually true is at most α. In other words, the false negative rate is constrained by the user-specified upper limit α. Likewise the probability of accepting H_0 given that H_1 is true should not exceed the user-specified requirement on false positive rate β. These constraints are also captured in the formal definition of the sequential decision problem (5.7). In the original work the problem of measurement ordering is not considered and all measurements are assumed to be independent and identically distributed (*i.i.d.*).

Suppose that before taking the first observation x_i there exists an a priori probability that H_0 is true. Denote the value of this a priori probability by $P(H_0)$. Likewise the probability $P(H_1) = 1 - P(H_0)$ denotes the a priori probability of H_1 being true. As is common in statistical experiments, the probabilities of H_0 and H_1 being true change when making additional observations during the experiment. After taking t observations the probabilities of H_0 or H_1 being true change based on the outcome of the observations. The a posteriori probabilities of the hypotheses being true after t observations are denoted by $P\left(H_0 \mid x_1, \ldots, x_t\right)$ and $P\left(H_1 \mid x_1, \ldots, x_t\right)$ respectively.

Starting from these fundamental probabilities, it can be shown using Bayes' rule that the optimal strategy is given by the sequential probability ratio test first derived by Wald [90]. In Appendix A we give mathematical background behind the ratio test and present a condensed proof of the optimality. However the full proof is hard and the interested reader is referred to Wald [90].

The **Sequential Probability Ratio Test (SPRT)** is given by:

$$S_t^\dagger = \begin{cases} \text{Accept } H_0 & \text{if } R_t \leq B \\ \text{Accept } H_1 & \text{if } R_t \geq A \\ \sharp & \text{if } B < R_t < A \end{cases} \tag{5.8}$$

Where R_t denotes the *likelihood ratio* between the two classes. In other words it denotes the ratio between the conditional probability densities under the hypotheses H_0 and H_1.

$$\boxed{R_t(x) = \frac{P\left(x_1, \ldots, x_t \mid H_1\right)}{P\left(x_1, \ldots, x_t \mid H_0\right)}} \tag{5.9}$$

The likelihood ratio $R_t(x)$ decreases if the probability H_0 being true becomes larger. Parameters A and B correspond directly to the final false positive and false negative detection rates. The parameters A and B are set according to the requirements on the error of the first and second type; in his original work Wald determines [90] upper and lower bounds on A and B in terms of α and β and suggests using these bounds as practical setting for the sequential probability ratio test:

$$A' = \frac{1-\beta}{\alpha}, \qquad B' = \frac{\beta}{1-\alpha} \tag{5.10}$$

Note that the decision strategy discussed above implicitly minimizes the expected classification time while asserting that the error rate requirements are satisfied. Two important assumptions for the sequential test are that the measurements to be selected are ordered a priori and that the measurements are class-conditionally independent or their joint probability density functions are known. In the context of object detection these assumptions clearly do not hold; measurements are not independent and we do not know the underlying joint-probability density functions of the classes. However the ratio test can still be used if the likelihood ratio $R_t(x)$ can be estimated.

This is were we shift our attention from the purely theoretical framework proposed by Wald [90] to the practical application in the context of boosting and object detection by Šochman *et al.* [79] who introduce WaldBoost. In this more recent work the authors propose using the AdaBoost algorithm for class-conditional probability density estimation for estimating the likelihood ratio (5.9). Since the AdaBoost algorithm in its original form already serves as measurement selector for the statistical experiment $\{x_1, x_2, \ldots, x_t\}$ by feature selection, the additional purpose of estimating the likelihood ratio forms a beautiful framework to build a high-speed classifier. It the following sections we describe

the idea behind WaldBoost and also focus on the density estimation process which forms an important part of the framework.

5.4 WaldBoost

WaldBoost is a soft-cascade algorithm that integrates Wald's SPRT into a boosting framework for fast classification. Put differently, the goal of WaldBoost is to minimize the average decision time while guaranteeing the required false positive and false negative rates on the training set. Since its introduction by Šochman *et al.* [79], the algorithm has been found useful in various computer vision applications [42, 39, 41]. In the context of object detection the use of WaldBoost is motivated by the fact that the majority of image subwindows does not contain the object of interest. Therefore, early rejection of these windows will cause a dramatic reduction in overall detection time.

The foremost contribution of the work by Šochman *et al.* is the insight that AdaBoost features (weak classifiers) are in fact an ordered sequence of measurements that can be made on an object. Hence the classification problem using AdaBoost naturally expresses as sequential decision process in which the weak classifiers h_t serve as measurements. This insight also shows that the sequential probability ratio test can be used for making a decision at each stage, *i.e.* determine the next step after evaluating a weak classifier and updating the trace $H_t(x)$. Taking into account that observations x in the likelihood ratio (5.11) correspond to weak learners $h^{(i)}(x)$, the likelihood ratio in the context of object detection using AdaBoost can be written as:

$$R_t(x) = \frac{P(h^{(1)}(x), \ldots, h^{(t)}(x) \mid y = -1)}{P(h^{(1)}(x), \ldots, h^{(t)}(x) \mid y = +1)} \tag{5.11}$$

Where we now explicitly write class labels $y = \pm 1$ instead of hypotheses H_0 and H_1. Wald did not consider the optimal ordering of measurements, since he assumes that all measurements are *i.i.d.* and therefore the order does not matter. However outputs of the weak classifiers given in (5.11) cannot be treated as statistically independent. Fortunately for dependent measurements, the sequential probability ratio test can still be used if the likelihood ratio can be estimated with sufficient accuracy. Therefore we need to find a method of approximating this likelihood ratio.

5.4.1 One-Dimensional Likelihood Ratio Estimation

Direct estimation of $R_t(x)$ involves a high-dimensional density estimation in t dimensions. Such high-dimensional density estimates quickly become infeasible from a computational perspective, hence this is not possible in our context for which t can become larger than 1000. To avoid the computation of $R_t(x)$ involving high-dimensional density estimation, Šochman *et al.* [79] propose a relatively simple approximation strategy that simplifies the density estimation task to a single dimension. This is done by projecting the t-dimensional space constituted by the weak classifiers onto the $1D$ dimension of the strong classifier

cascade score $H_t(x)$. Recall that the cascade score $H_t(x)$ given in is defined as weighted sum of the weak learner outputs (see Equation (5.1)). Therefore the dimension reduction mapping is performed by summation of weighted weak learner outputs $\left\{h^{(1)}, \ldots, h^{(t)}\right\}$. Using these insights, it is shown [79] that an estimate of the likelihood ratio is given by:

$$\widehat{R}_t(x) = \frac{P(H_t(x) \mid y = -1)}{P(H_t(x) \mid y = +1)} \tag{5.12}$$

Justification of this approximation is given by the following reasoning. Discrete AdaBoost can be interpreted as stage-wise estimation procedure for fitting an additive logistic regression model [74]. By working out the minimization of the loss function, it is shown by Friedman *et al.* [32] that choosing a weak classifier according to AdaBoost causes the strong classifier score to converge asymptotically to the following expression related to the likelihood ratio:

$$\lim_{T \to \infty} H_T(x) = \widetilde{H}(x) = \frac{1}{2} \log \frac{P\left(y = +1 \mid x\right)}{P\left(y = -1 \mid x\right)} \tag{5.13}$$

Note that $\widetilde{H}(x)$ is proportional to the likelihood ratio (5.11). Using Bayes' rule we can rewrite this in the following way.

$$\begin{aligned}
\widetilde{H}(x) &= \frac{1}{2} \log \frac{P(y = +1 \mid x)}{P(y = -1 \mid x)} \\
&= \frac{1}{2} \log \frac{P(x \mid y = +1)P(y = +1)}{P(x \mid y = -1)P(y = -1)} \\
&= \frac{1}{2} \log \frac{P(x \mid y = +1)}{P(x \mid y = -1)} + \frac{1}{2} \log \frac{P(y = +1)}{P(y = -1)} \\
&= -\frac{1}{2} \log R_t + \frac{1}{2} \log \frac{P(y = +1)}{P(y = -1)}
\end{aligned} \tag{5.14}$$

This final relationship (5.14) shows that for the asymptotic case, the strong classifier is directly related to the likelihood ratio. More specifically, all points for which the likelihood ratio $R_t(x)$ is the same, the classifier maps the output to the same value $\widetilde{H}(x)$. Based on these observations we see that the estimate (5.12) is an exact result in the asymptotic case, *i.e.* $t \to \infty$. For the non-asymptotic case Šochman *et al.* [79] argue that the same relationship between $H_t(x)$ and $\widehat{R}_t(x)$ also holds approximately and therefore the likelihood ratio estimate holds also for $t \ll \infty$.

Numerical evaluation of the likelihood ratio estimate (5.12) requires one-dimensional class-conditional density estimates of $P(H_t(x) \mid y = \pm 1)$. In Subsection 5.4.2.1 we describe the process of estimating the likelihood ratio using non-parametric density estimation, but for now lets assume that we have accurate estimate of $\widehat{R}_t(x)$. Given the likelihood ratio and requirements on the error rates, the SPRT can be applied directly. In the context of measurements from AdaBoost the error bounds A and B are directly related to the

stage rejection thresholds $\theta_A^{(t)}$ and $\theta_B^{(t)}$ respectively. The sequential probability ratio test for a two-class decision problem with AdaBoost as measurement selection then becomes:

$$S_t^\dagger = \begin{cases} +1 & \text{if } H_t(x) \geq \theta_B^{(t)} \\ -1 & \text{if } H_t(x) \leq \theta_A^{(t)} \\ \sharp & \text{if } \theta_A^{(t)} < H_t(x) < \theta_B^{(t)} \end{cases} \tag{5.15}$$

Note that the inequalities are inverted compared to (5.8) since $H_t(x)$ is proportional to $-\widehat{R}_t(x)$ which can be seen from (5.14). In words this strategy simply denotes that an object x can be classified before reaching the last weak classifier based on its current trace $H_t(x)$. While a very simple strategy, the difficulty is in deriving the bounds $\theta_A^{(t)}$ and $\theta_B^{(t)}$ using Equation (5.10). This strategy is an important result that is used in our implementation for both training and evaluation – details are given later in this chapter. The decision making process is evaluated at every stage for the classification process and reflects the minimization of expected evaluation time per object. The samples are labeled as soon as possible within the error rate constraints. We continue our discussion with describing the method in which stage rejection thresholds can be determined which involves density estimation.

5.4.2 Estimation of Stage Thresholds θ_A and θ_B

The WaldBoost algorithm depends on an accurate estimation of the likelihood ratio $\widehat{R}_t(x)$. In the original paper [79] the authors suggest splitting the dataset into a training and validation subset for obtaining an unbiased density estimate. The probability densities $P(H_t(x) \mid y = \pm 1)$ are not estimated directly from the training set but rather are estimated from the independent validation set which is not used for training the classifier. Estimating the densities using this validation set reduces the estimation bias to a minimum when compared with estimation directly from the training set. However, as is discussed later in this chapter, the rejection thresholds are used for pruning samples from the training data, hence this method is not completely independent and unbiased. The actual density estimation itself is performed using non-parametric density estimation with Parzen windows.

5.4.2.1 Non-Parametric Density Estimation

Non-parametric density estimation is a family of methods for estimating unknown probability density distributions using only a finite set of samples. In the context of WaldBoost the unknown density $P(H_t(x) \mid y = \pm 1)$ be can estimated using the finite set of samples available in the independent validation set. Since the probability density function and its parameters are unknown, the method at hand is non-parametric density estimation. Higher dimensional density estimation can be seen as a generalization of histogram density estimation. Before discussing density estimation, we refer the reader to Figure 5.2 illustrating the class-conditional probability densities.

For the simple case of one-dimensional density estimation we can divide the x-axis into bins of length h. We are given a total of N samples that can be used to estimate the density function. Let k_n be the number of samples inside a bin, then the probability can simply be approximated by the frequency ratio $P \approx k_n/N$. Assuming the corresponding density function is constant over the range of one bin, the probability density function can be approximated by [84]:

$$\widehat{f}(x) = \frac{1}{h}\frac{k_n}{N}, \qquad |x - \widehat{x}| \leq \frac{h}{2} \tag{5.16}$$

where $\widehat{x}$ is the midpoint of the bin. It can be shown that $\widehat{f}(x)$ converges to the true value of $f(x)$ as $N \to \infty$ given the bin size is decreased for increasing N [84]. The number of samples N must be "large enough" and the bin size h must he "small enough". Choosing these parameters is not trivial in practice but fortunately there are rules of thumb that we will discuss in a moment. The one-dimensional histogram approach can be generalized to the multi-dimensional case by definition of *Parzen windows* [62]. For the multi-dimensional case the bins with a certain width become hypercubes with a given volume $V = h^D$ where h now denotes the length of the hypercube centered around a point x origin and D the number of dimensions. To explicitly indicate multidimensional density estimation, we denote input as vector $\mathbf{x}$. This reformulation allows us to generalize Equation (5.16) to the multi-dimensional version of the density estimate:

$$\widehat{f}_{KDE}(\mathbf{x}) = \frac{1}{Nh^D}\sum_{i=1}^{N} K\left(\frac{\mathbf{x_i} - \mathbf{x}}{h}\right). \tag{5.17}$$

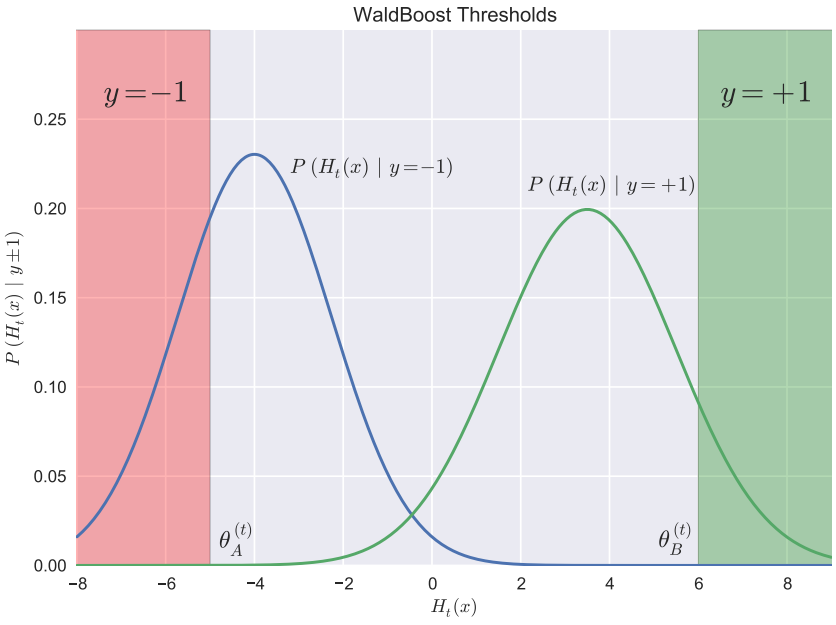


Figure 5.2: Class-conditional probability density functions $P(H_t(x) \mid y = \pm 1)$ and illustration of the WaldBoost thresholds θ_A and θ_B determined using the sequential probability ratio test.

The functions $K(\cdot)$ are smooth functions that are positive and integrate to one, *i.e.* $\int K(\mathbf{x})\, d\mathbf{x} = 1$ and $K(\mathbf{x}) \geq 0$. These smooth functions are commonly referred to as kernels or Parzen windows [62]. Large values of h result in smooth estimates but generally the approximation accuracy is low. On the other hand, smaller values of h typically result in spiky but more but are more accurate estimates. The approximation accuracy increases with smaller values of h and larger numbers of samples. In practice the Gaussian kernel is commonly utilized for density estimation. Substitution of the Gaussian kernel for $K(\cdot)$ into Equation (5.17) results in the density given by:

$$\widehat{f}(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{(2\pi)^{D/2} h^D} \exp\left(-\frac{(\mathbf{x} - \mathbf{x_i})^T (\mathbf{x} - \mathbf{x_i})}{2h^2} \right). \tag{5.18}$$

In words this equation expresses that each observation $\mathbf{x}_i$ contributes to the density estimate, and that importance of contribution is Gaussian function of the distance between the query point $\mathbf{x}$ and the sample $\mathbf{x}_i$. After fixing the kernel function, one also needs to set the kernel bandwidth h. For estimating univariate data using Gaussian kernels it can be shown [78] that the optimal choice for bandwidth is given by:

$$h = \left(\frac{4\widehat{\sigma}^5}{3N} \right)^{\frac{1}{5}} \approx 1.06 \cdot \widehat{\sigma} \cdot N^{-1/5}. \tag{5.19}$$

Where $\widehat{\sigma}$ denotes the standard deviation computed over the N samples that are available for density estimation, also referred to as our validation dataset. This rule of thumb for setting the bandwidth is sometimes referred to as Silverman's rule of thumb [78]. All density estimations throughout this work are estimated using Gaussian kernels with bandwidth set according to the rule of thumb. Now that we have described non-parametric density estimation, we can estimate $P(H_t(x) \mid y = \pm 1)$ from an independent validation set and then compute the likelihood ratio estimate $\widehat{R}_t(x)$ from which the stage thresholds can be computed directly using the SPRT.

$$P\left(H'_t \mid y = +1\right) = \frac{1}{N_+} \sum_{i=1}^{N_+} \frac{1}{\sqrt{2\pi} h_+} \exp\left(-\frac{H'_t - H_t(x_i)}{2h_+^2} \right) \tag{5.20}$$

$$P\left(H'_t \mid y = -1\right) = \frac{1}{N_-} \sum_{i=1}^{N_-} \frac{1}{\sqrt{2\pi} h_-} \exp\left(-\frac{H'_t - H_t(x_i)}{2h_-^2} \right). \tag{5.21}$$

Where H'_t denotes a "query" value for which the probability is evaluated. The summations are over the positive and negative samples respectively, indicated with N_+ and N_-. Note that the kernel bandwidths are different since the standard deviation and numbers of samples are different in Equation (5.19) for both classes. This concludes our low-level discussion on WaldBoost allowing us to put all things together in defining the actual training and classification process.

5.4.3 WaldBoost Training

The WaldBoost *training* algorithm is summarized in Algorithm 3 [79]. The training process is almost identical to Discrete AdaBoost but with the important addition of learning stage thresholds $\theta_A^{(t)}$ and $\theta_B^{(t)}$. Prior to starting the training process, the system requires user-specified parameters α and β. These are then used to initialize the bounds A and B according to Equation (5.10). The training process runs in a loop identical to Discrete AdaBoost. After searching the best weak classifier for each stage, the total cascade score $H_t(x)$ is updated for all samples in the *validation* set. The next step is to estimate the class-conditional probability densities using this independent validation set. Given the density estimates computed using the method discussed in the previous section, the likelihood ratio $\widehat{R}_t(x)$ can be computed using Equation (5.12). Since no analytical expression of the probability densities is available, the densities are evaluated for a fixed number of evenly spaced trace values (*e.g.* 1000 query points).

After computing the likelihood ratios, computation of the stage thresholds is straightforward since we can apply the SPRT directly (given in Equation (5.8)). For our implementation we assume that the likelihood ratios are ordered in increasing trace value; then we approach from the "left" and set the stage thresholds as follows (x corresponds to H_t):

$$\theta_A^{(t)} = \arg\max_x \left\{ \widehat{R}_t(x) \geq A \right\} \tag{5.22}$$

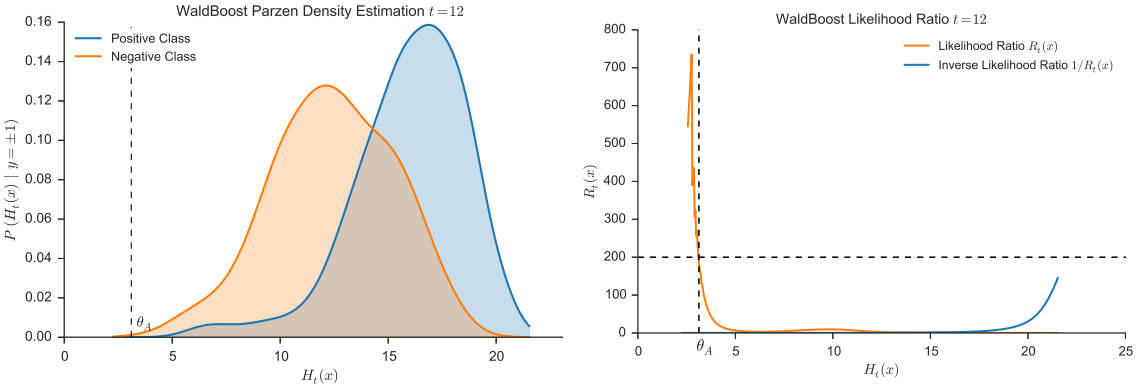
$$\theta_B^{(t)} = \arg\min_x \left\{ \widehat{R}_t(x) \leq B \right\} \tag{5.23}$$

In Figure 5.3 we illustrate the process of determining the thresholds. The subfigure on the right illustrates that we approach $H_t(x)$ from the left and update $\theta_A^{(t)}$ until the likelihood ratio estimate drops below the bound A. Note that for this particular training process $\beta = 0$, meaning that no threshold $\theta_B^{(t)}$ is set for making early decision on the positive label of an object.

As a final step for each training iteration *bootstrapping* can be utilized for exploring a large amount of training examples without excessive memory usage. After a weak classifier is trained and stage thresholds are determined, usually the "easy" samples with a trace score exceeding one of the two stage thresholds are removed from the training set and new training data is randomly sampled into the set. Learning with bootstrapping introduces maximum variety in the training set while keeping memory requirements acceptable since not all training samples are loaded into memory.

5.4.4 WaldBoost Classification

The structure of WaldBoost *classification* is summarized in Algorithm 4. Again the classification process is similar to AdaBoost classification. The classification executes the sequential probability ratio test via the strong classifier H_T with the sequence of thresholds $\theta_A^{(t)}$ and $\theta_B^{(t)}$ determined for each stage during the training phase. If at any



(a) Class-conditional probabilities

(b) Likelihood ratio estimate

Figure 5.3: WaldBoost threshold estimation $\theta_A^{(t)}$ and $\theta_B^{(t)}$. The training parameters are set to $\alpha = 0.005$ and $\beta = 0$. The horizontal line indicates the bound $A = 200$ which is used for computing threshold $\theta_A^{(t)}$. Since $\beta = 0$ no threshold $\theta_B^{(t)}$ is determined.

given stage H_t exceeds the corresponding thresholds, a decision is made according to the SPRT given in Equation (5.15). The sequential classification task continues until a decision is made or the final stage T in the cascade is reached. If a decision is not made within T stages, the input is classified by thresholding H_T on a value γ which can be user-specified or set according to AdaBoost threshold (4.15). Note that this threshold can also be used to control the false positive and false negative rate in the application and for generating precision-recall curves.

5.4.5 WaldBoost Discussion

The WaldBoost algorithm is particularly useful for classification tasks that have strong requirements on classification speed. Given the fact that many computer vision tasks should operate in real time, WaldBoost is of particular interested to these applications. In Chapter 7 we perform experiments and show that the classification speedup offered by WaldBoost is indeed imperative for achieving real time performance. In addition to the classification speedup, the WaldBoost training process (Algorithm 3) has two additional advantages for object detection applications compared to traditional AdaBoost learning [79].

Advantage 1: Learning with Bootstrapping. The first advantage is learning with bootstrapping as briefly described in Section 5.4.3. This improvement is motivated by the fact that typically in object detection the two classes are highly unbalanced, *i.e.* the positive class (object of interest) is relatively small compared to the background class. The positive class is hard to collect and typically requires manual annotation, while examples for the background class are widely available and easy to collect automatically. Not only differ these two classes in size, but also with respect to "information diversity". The positive class is compact in diversity in comparison to the highly diverse background class. As a result the huge and highly complex background class cannot be represented

Algorithm 3 WaldBoost Training [79]

Require: Training examples $\{(x_i, y_i), \ldots, (x_n, y_n)\}$.

Require: Validation examples $\{(x_i, y_i), \ldots, (x_m, y_m)\}$ for density estimation.

Require: Requirements on final false negative rate α and final false positive rate β.

Require: Number of weak classifiers T to train.

1: Initialize sample weights for positive and negative samples to $w_1 = \frac{1}{2n_+}$ and $w_1 = \frac{1}{2n_-}$, where n_+ and n_- represent the numbers of positive and negative samples respectively.

2: Compute the following thresholds from the user-specified error requirements:

$$A = \frac{1-\beta}{\alpha}, \qquad B = \frac{\beta}{1-\alpha}. \tag{5.24}$$

3: **for** training stages $t = 1, 2, \ldots T$ **do**

4: Find the best weak classifier h_t using AdaBoost (Algorithm 1) on training examples.

5: Update trace $H_t(x)$ for all training *and* validation samples using weak classifier h_t

6: Estimate $P(H_t(x) \mid y = \pm 1)$ over validation samples using (5.20) and (5.21).

7: Using the density estimates, compute the likelihood ratio R_t according to:

$$\widehat{R}_t(x) = \frac{P(H_t(x) \mid y = -1)}{P(H_t(x) \mid y = +1)} \tag{5.25}$$

8: Find stage threshold $\theta_A^{(t)}$ and $\theta_B^{(t)}$ using (5.22) and (5.23).

9: Remove samples from the training set for which $H_t \geq \theta_B^{(t)}$ or $H_t \leq \theta_A^{(t)}$.

10: Sample new data into the training set and renormalize the sample weights.

11: **end for**

12: Output the **strong classifier** H_T and stage thresholds $\theta_A^{(t)}$ and $\theta_B^{(t)}$.

by a small set of training examples. Therefore learning with bootstrapping offered by WaldBoost is excellent for exploring the largest possible subspace of the background class while keeping most of the positive objects in the training set.

Advantage 2: Requirements on Error Rates. The second property that makes WaldBoost an attractive choice for object detection compared to AdaBoost and different soft-cascade approaches, are the requirements on the error of the first and second kind. The error of the first kind (missed object of interest) is typically considered more serious than an error of the second kind (falsely classified object of interest). For such requirements WaldBoost is perfect since its two input parameters α and β are the constraints on these error rates. Therefore the system requirements are easily set prior to the training process.

Taking into account the properties described previously, in general the ideal training setting would be to require zero false negative rate and the smallest possible false positive rate. With these properties in mind, the parameters displayed in Table 5.1 are established

Algorithm 4 WaldBoost Classification [79]

Require: Strong classifier H_T consisting of T weak classifiers $h^{(t)}$.
Require: WaldBoost stage rejection thresholds $\theta_A^{(t)}$ and $\theta_B^{(t)}$.
Require: Final cascade threshold γ.
Require: An object or image subwindow x to classify.

1: **for** stages $t = 1, 2, \ldots T$ **do**
2: Predict the label of x according to weak classifier $h^{(t)}(x)$.
3: Update $H_t(x)$ according to Equation (5.1) if predicted label $y = +1$.
4: If $H_t(x) \geq \theta_B^{(t)}$, classify object to class +1 and terminate.
5: If $H_t(x) \leq \theta_A^{(t)}$, classify object to class −1 and terminate.
6: **end for**

7: Output label +1 if $H_t(x) \geq \gamma$, otherwise return −1.

throughout this thesis. In this setting, the upper bounds A and B defined in Equation (5.10) reduce to:

$$A = \frac{1-0}{\alpha} = \frac{1}{\alpha} \tag{5.26}$$

$$B = \frac{0}{1-\alpha} = 0 \tag{5.27}$$

Substituting these results into the SPRT strategy (5.8) we achieve the desired result. Since the likelihood ratio R_t is always positive, the algorithm will never classify a sample to the positive class during the decision process. The only allowed decision is the classification to the background class. For the training process this means that positive objects are never pruned while background samples x with a trace $H_t(x) < \theta_A^{(t)}$ are removed from the active training set. Such initialization thus leads to an exploration of the background class while training on a constant set of positive samples. Moreover since α corresponds to the false negative rate, by setting these parameters and Wald's proof on the bounds A and B, we are assured that the detection rate of the final classifier is guaranteed to be $1 - \alpha$ on the training set. The false positive rate is progressively reduced by each training iteration, *i.e.* addition of a weak classifier [79].

Table 5.1: Typical settings for WaldBoost classification in object detection context.

Parameter	Typical Value	Description
α	0.005	At most 0.5% of objects of interest can be rejected in training.
β	0	No background samples can be classified as object.

5.5 Chapter Summary

In this chapter, two-class classification of image subwindows was formulated in a framework for sequential decision making. WaldBoost explicitly formulates the problem in terms of trade-off between detection quality and time, and uses Wald's sequential probability ratio test for learning stage thresholds. While the original work by Wald [90] only is applicable for independent measurements during the decision process, Šochman *et al.* [79] have generalized this ideas for dependent measurements. In the proposed WaldBoost algorithm the measurements are "selected" by AdaBoost and the probability density functions are estimated using a one-dimensional projection onto the cascade score of the strong classifier. To reduce the effect of biased estimates, an independent validation set is adopted for non-parametric density estimation. In Chapter 7 we perform experiments on WaldBoost and show that its implementation results in a significant decrease in average classification time per subwindow in the image. As a result the fraction of time spent on computing the image pyramid and extracting image channels for each scale increases significantly (Figure 5.4). This motivates us to improve multiscale handling of our detector in the next chapter.

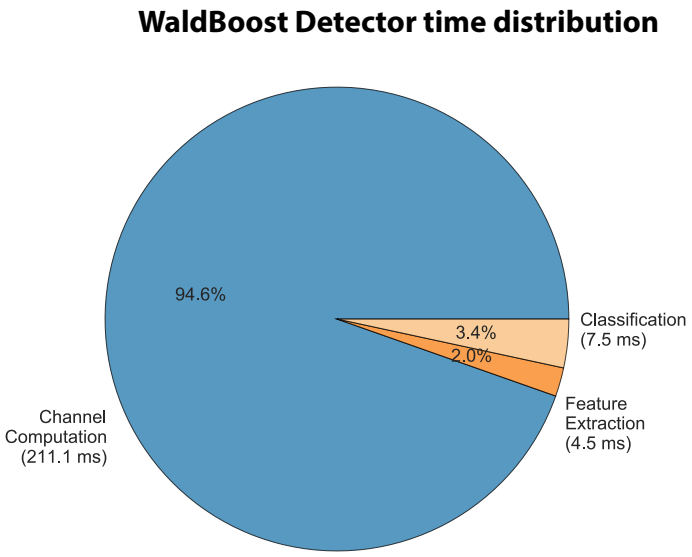


Figure 5.4: Per-component time contribution WaldBoost detector.

6

Multiscale Feature Approximations

In the previous chapter we have focused on speeding up *classification* for subwindows in an image. WaldBoost offers significant speedup in classification time, and as we will see in Chapter 7, without a significant decrease in detection quality. On average the time spent on feature evaluation for an entire image is reduced dramatically since the majority of image windows is rejected after only one or two observations in the decision process. The baseline detector introduced in Chapter 4 was extended with the WaldBoost decision process for speed improvements. At this stage our object detector uses WaldBoost for classification and the fast GPU implementation for computing channel images. In this chapter the focus is on improving multiscale handling of our detector.

As was briefly touched upon in Chapter 2, multiscale object detections are typically performed using the following approach: to detect objects larger than the template size the image is downsampled, whereas for detecting smaller objects the image is upsampled. At each scale the features are recomputed and single-scale detection is applied by applying the template in sliding-window fashion. Experiments in [10] and [13] show that classification performance drops significantly when evaluating less than 8 image scales per octave. With an octave we indicate the interval between one scale and another with half its value. For detections over four image octaves this results in an image pyramid consisting of at least 32 levels. Benchmarking our detector at this stage shows that most of the time is spent on rescaling the input images and computing the channels for each scale. This observation is not surprising: even with highly optimized code, the construction of the image pyramid and computing the features at every scale can take more than one second for 640×480 images on a modern desktop computer [24].

Clearly the feature extraction process for all these scales is a computational extensive task and motivates us to find ways to overcome this process. In this chapter we focus on *multiscale detections* without the explicit need for computing the image pyramid and extracting features at all scales. We study scaling properties of image statistics and the

application to feature pyramids in the context of object detection. The structure of this chapter is as follows. First we briefly introduce the reader to the concept of scale space and scale invariance. Then we discuss the scaling behavior for histograms of oriented gradients, which serves as an example for introducing the idea proposed in the sections thereafter. This idea is then generalized in a powerful relationship between image statistics over various scales. In the last section of this chapter, we describe a framework in which this relationship is used for rescaling channel features to be adopted in a framework for fast object detection without image rescaling.

6.1 Introduction to Multiscale Analysis

In preceding chapters we have extensively studied the object detection framework by Viola and Jones [88]. VJ showed the benefits of "scaling the features, not the images", resulting in a very fast face detector. In addition to being shift invariant, Haar features are also scale invariant but more importantly the scale invariance property can easily be exploited for multiscale object detection without rescaling the image. Because Haar features are evaluated over the luminosity channel, the features can be evaluated by simply rescaling the feature area, changing its position and modifying the detection threshold to compensate for the change in area. Eliminating the need for explicit image rescaling at runtime is reflected in high detection speeds achieved by VJ face detector. Unfortunately the Haar features used by VJ are an exception when it comes to simple feature rescaling for multiscale detections. For example the scale invariance property can not be exploited directly for HOG-like features due to blurring and interpolation effects [3]. Motivated by the fact that HOG features are popular and feature extraction for each scale is expensive, researcher have studied the scaling behavior of HOG features to be considered as extension for fast object detectors. Since our feature representation of channel features can be considered an approximation of HOG features, these studies are also interesting with respect to our work. The possibility to approximate HOG features on various image scales without having to rescale the image explicitly is tempting.

Intuitively it is possible to estimate various image statistics at scales different from the original image. This idea is widely studied in *scale space* literature, most notably by Lindeberg [50, 34], Koenderink [44, 45] and Florack [30, 29]. These works describe the relationship between image structure over various scales, *i.e.* the scale space. A simple example described by Lindeberg [51] is that of a branch of a tree. It makes sense only to study the branch of a tree at a scale of centimeters to meters. At nanometer scale it makes more sense to talk about the molecules in the tree while at kilometer scale only the forest in which the tree grows is relevant. The fact that objects in the world appear in different ways depending on the scale of observation has important implications when studying images at various scales. Scale space theory is a general framework that has been developed in the computer vision community for representing image data and the multiscale nature of it at the early stages in the chain of visual processing [51]. While

the works by Lindeberg *et al.* are rather theoretical, their ideas have successfully been embraced in multiscale computer vision frameworks.

There is an important difference between the nature of *mathematical* objects and *physical* objects. In mathematics objects have no scale: no metrical units are involved and we are familiar with the notion of infinitely small points. In physics, on the other hand, objects live on a range of scales. We need an instrument to do an observation (*e.g.* our eyes or a camera) and we are limited to our observations by the scale range imposed by the instrument [67]. This fundamental difference is of great importance in image processing where we deal with discrete signals measured from the physical world. In mathematics we consider the smallest distance by taking the limit to zero, but in physics this reduces to a finite separation distance, *i.e.* the sampling distance. Therefore we should be aware of dealing with concepts such as differentiation of discrete signals and subpixel accuracies. Purely mathematically speaking we can study the scale behavior of some function $f(x)$ under rescaling of variable x. In other words, we are interested in the shape of $f(kx)$ for some scale factor k. We call $f(x)$ *scale invariant* if the following relationship holds for all scales k:

$$f(kx) = k^{\lambda} f(x). \tag{6.1}$$

For some exponent λ characteristic for the function. This relationship is equivalent to saying $f(x)$ is a homogeneous function. A closely related concept is *self-similarity*, in which a function or object is scale invariant only for a discrete subset of scales. This concept features close resemblance with *fractal statistics*, describing the general concept of self-similarity over a number of scales.

As we will see in later in this chapter, natural images are known to have scale invariant statistics [95]. In other words, the marginal distribution of statistics of natural images remain unchanged if the images are rescaled [81]. However the concept of scale invariance is a fragile property that one typically does not observe exactly, even more so because of the scale range in the physical world. Instead, what tends to be observed in practice is an approximation of the scale invariance property defined in (6.1). In Section 6.3 we study these approximations by collecting a large amount of image statistics.

As discussed in the beginning of this section, HOG features cannot be approximated directly for different scales than at which the image is available. On the contrary, this is possible for Haar features since in this case features are extracted as differences over image regions limited to the luminosity channel. Most features are *not* scale invariant, including gradient orientation histograms. Due to the discrete binning to a number of orientation bins, the feature response changes dramatically when rescaling the image, *i.e.* performing explicit multiscale analysis. It seems like we are facing a dilemma: (1) choose between highly informative features such as histograms of oriented gradients, but explicitly perform detections on all scales by rescaling the image, or (2) use simpler descriptors such as Haar features which allow for simply rescaling the features resulting in fast multiscale detections. This dilemma was studied by Dollár *et al.* [15, 13] who suggests a method for rescaling HOG features by approximation. This compelling idea

opens doors for fast multiscale objects detections based on HOG descriptors. In the following sections we discuss these ideas for approximating HOG features and working towards integrating this aspect in our detector. We begin our discussion with the example of rescaling behavior of histograms of orientated gradients.

6.2 Scale Invariance of Histogram of Oriented Gradients

From a recent survey paper [4] on pedestrian detection we recognize that many of the high-performing object detectors are using a HOG-like feature representation. In other words, histograms of oriented gradients seem to find the right balance between capturing edge and local shape information and the computational efficiency with which they can be extracted. Our discussion in Section 4.1 has demonstrated that channel features can be used for approximating HOG features over large regions. Just like channel features over gradient orientation channels can not be rescaled in a straightforward way due to the discrete binning of orientations which are noisy on a per-pixel level. However intuition tells us that it is possible to derive an approximation for histograms of orientated gradients at different image scales. In this section we follow up on our intuition and study the behavior of histograms of oriented gradients over image scales.

Gradient histograms measure the distribution of the gradient orientations within an image. Let $I(x, y)$ be the input image and $\partial I/\partial x$ and $\partial I/\partial y$ the discrete derivatives in both directions. The gradient magnitude $M(i, j)^2$ and gradient orientation $O(i, j)$ are defined by:

$$M(i,j)^2 = \frac{\partial I}{\partial x}(i,j)^2 + \frac{\partial I}{\partial y}(i,j)^2 \tag{6.2}$$

$$O(i,j) = \arctan\left(\frac{\partial I}{\partial y}(i,j) \middle/ \frac{\partial I}{\partial x}(i,j)\right) \tag{6.3}$$

respectively. To compute the gradient histogram of an image, each pixel casts a vote, weighted by its gradient magnitude for the bin corresponding the orientation [15]. The gradient orientation $O(i, j)$ is quantized into Q gradient orientation bins. The q^{th} bin of the histogram is defined by

$$h_q = \sum_{i,j} M(i,j)\mathbb{I}\left[O(i,j) = q\right]. \tag{6.4}$$

Where $\mathbb{I}$ is the *indicator function* representing that $O(i, j)$ is within the q^{th} bin. Note that our work uses six different orientation bins illustrated in Figure 4.2. The following relationships not only hold for global histograms but are also applicable to local histograms which follows from changing the set of pixels i and j to a local region. In practice the image gradients can be computed using a variety of techniques, for example by convolving the image with horizontal and vertical derivatives of a Gaussian [83, 10]. As motivated in Section 4.4.1 we use simple 1D filter kernels for computing the gradients.

In the following two subsections we discuss the relationship between gradient histograms computed for an original image and an *upsampled* or *downsampled* versions of the image. These operations are fundamentally different in nature: downscaling removes structure since the number of pixels is decreased. Upscaling preserves structure since the content in the image does not change and more pixels are "available" for storing information. Therefore we describe the cases of upscaling and downscaling independently in the following two subsections.

6.2.1 Gradient Histograms in *Upsampled* Images

Let I be an *continuous* signal and I' denote an upsampled version of this signal by a factor k. The upsampled signal is k times "slower" than the original signal, *i.e.* $I'(x, y) = I(x/k, y/k)$. Using the chain rule for derivatives the corresponding derivatives of the upsampled signal are given by

$$\frac{\partial I'}{\partial x}(i, j) = \frac{1}{k}\frac{\partial I}{\partial x}(i/k, j/k), \qquad \frac{\partial I'}{\partial y}(i, j) = \frac{1}{k}\frac{\partial I}{\partial y}(i/k, j/k). \tag{6.5}$$

In words, these equations simply state the intuitive fact that the rate of change in the upsampled signal is k times slower than the rate of change in the original image. While this discussion focuses on continuous signals, the relationships also hold approximately for interpolated *discrete* signals. Let $M'(i, j) \approx \frac{1}{k}M(\lceil i/k \rceil, \lceil j/k \rceil)$ denote the gradient magnitude in an upsampled discrete image. We can derive the approximated relationship between the sum of gradients in the original image the corresponding sum in upscale image:

$$\begin{aligned}\sum_{i=1}^{kn}\sum_{j=1}^{km} M'(i, j) &\approx \sum_{i=1}^{kn}\sum_{j=1}^{km} \frac{1}{k}M(\lceil i/k \rceil, \lceil j/k \rceil) \\ &= k^2 \sum_{i=1}^{n}\sum_{i=1}^{m} \frac{1}{k}M(i, j) \\ &= k \sum_{i=1}^{n}\sum_{i=1}^{m} M(i, j)\end{aligned} \tag{6.6}$$

Where $kn = W'$ and $km = H'$ denote the width and height of the upscaled images respectively. This derivation shows that the sum of gradient magnitudes in the original and upsampled image should be related by about a factor k. Switching back to our notation for continuous signals, we can make a similar analysis for gradient orientations, since the rate of change in both directions after rescaling remains the same:

$$\frac{\partial I'}{\partial x}(i, j) \Big/ \frac{\partial I'}{\partial y}(i, j) \approx \frac{\partial I}{\partial x}(i/k, j/k) \Big/ \frac{\partial I}{\partial y}(i/k, j/k) \tag{6.7}$$

According to the definition of the histogram bins in Equation (6.4), the above expression of the gradient approximation results in a simple relationship between the histogram

bins at two different scales by resampling factor k. The relationship between h_q computed over the original image I and h'_q computed over the scaled image I_k is simply given by:

$$h'_q \approx k h_q \tag{6.8}$$

This relationship suggests that we can approximate gradient histograms over image regions for upscaled images just by computing of the image at the original scale. The validity of this relationship is empirically confirmed in Section 6.2.3. As a final note we point out the the similarity of (6.8) to the formal definition of scale invariance as given in Equation (6.1).

6.2.2 Gradient Histograms in *Downsampled* Images

As mentioned before, structural information is lost in the process of downsampling an image. Therefore the relationship derived in the previous section is not expected to hold exactly. Downsampling only results in information loss of high frequency content, while low frequency content is preserved. Let I' now denote I *downsampled* by a factor k. Given this change in definition relationship (6.8) becomes $h'_q \approx h_q/k$. However since high frequency structure is removed in the downsampling process and consequently gradients are also removed we expect the value of the histogram bins in the downscaled image to be strictly lower than our original estimation. In other words, the expectation is that the relationship between gradient histograms in downsampled images satisfies [15]:

$$h'_q \leq h_q/k \tag{6.9}$$

The factor k depends on the interpolation algorithm used for downscaling. Experiments in [13] show behavior for nearest-neighbor, bilinear and bicubic interpolation schemes. Varying the interpolation algorithm does not have a major effect on the accuracy of the approximation, as long as the interpolation algorithm remains fixed. For all our experiments bilinear interpolation is used unless specified otherwise.

6.2.3 Experiments on Gradient Histograms in *Upsampled* and *Downsampled* Images

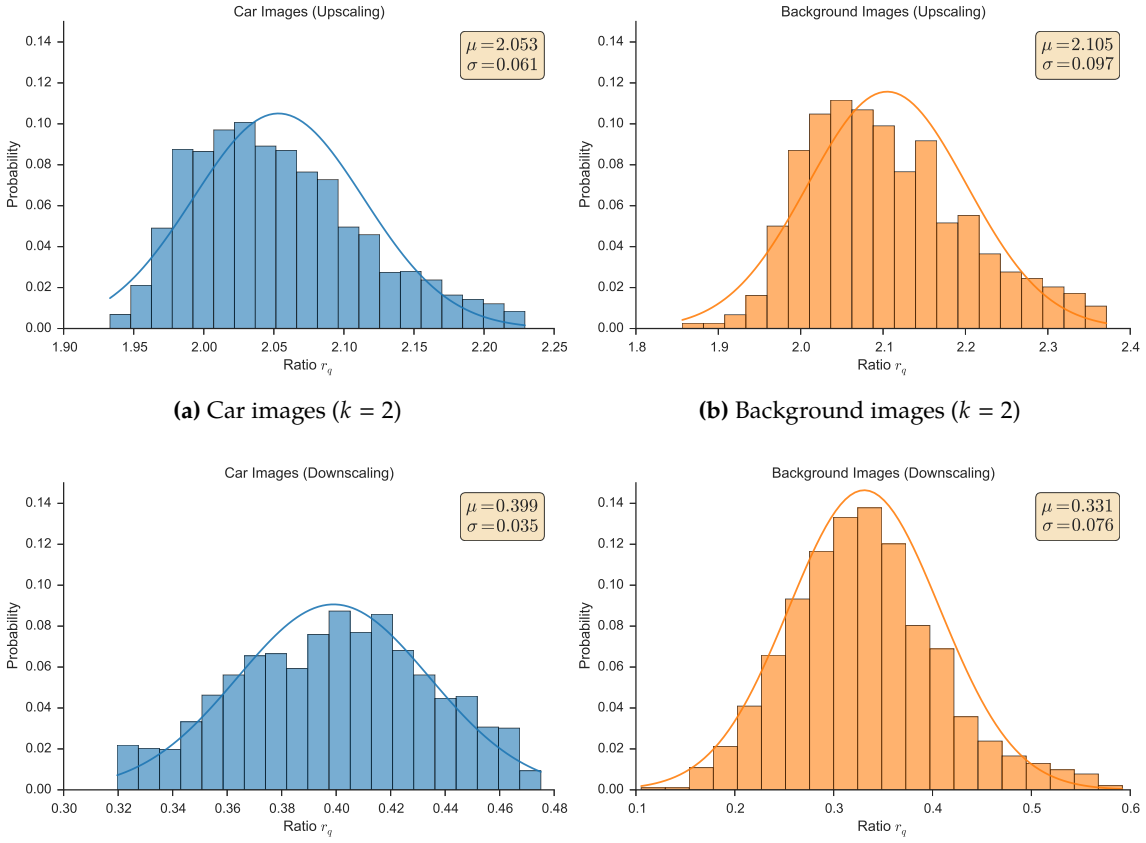
Dollár *et al.* [13, 15] perform experiments to verify the suggested relationships for upscaling and downscaling of the gradients expressed in Equations (6.8) and (6.9) respectively. While these relationships are expected to hold in general, the authors choose to validate them using experiments performed on a large set of pedestrian images. We report similar results by performing the same experiment on our TomTom dataset of rear-view car images. Our experiment is performed by taking two datasets: our car dataset consisting of 2000 images and the set of background samples extracted randomly from the original MoMa images. Initially the images are equally sized 80×80. We construct both an *upscaled* and *downscaled* version of the images by rescaling the images with factors $k = 2$ and $k = 0.5$. For all scales: original, downsampled and upsampled, the histograms of

oriented gradients are computed for $Q = 6$ orientation bins. For details on the orientation histogram binning, see Figure 4.2 and enclosing discussion. In order to measure the fidelity of the relationships we define the ratio $r_q = h'_q/h_q$. We compute this ratio for all images in our dataset and evaluate the expectation value and standard deviation to improve our understanding of how gradient orientation histograms behave in real world images over multiple scales.

Figure 6.1 (top row) displays the effect on the gradient orientation histogram after upscaling the image with a factor 2. In agreement with Equation (6.8) we report an average equal to the scaling factor, *i.e.* $\mu \approx 2$. One observation that can be made from this figure is that the estimation is less accurate for the set of background images compared to the car images. This effect is explained by the fact that background images can possibly contain little or irregular image structures, *e.g.* sky, road, trees. As this experiment shows, gradient histograms in original and upscaled image ensembles are related by a multiplicative constant roughly equal to the scale change between them. Downsampling results are plotted in Figure 6.1 (bottom row). The mean $\mu \approx 0.33$ is lower than the factor 0.5 that we might expect from the case of upsampling. However, the lower factor can be explained by the fact that downsampling causes loss of high frequency content [15]. However we also note that for background images in the case of downsampling there are bins for which $r_q > 0.5$. This can happen if due to the scaling and interpolation some per-pixel orientations are put into different bins. We emphasize that the scaling behavior is studied over a large number of images, and the approximation can be noisy for individual images.

So far we have limited ourselves to two scaling factors, however we are also interested in how these relationships hold over larger scale ranges. To generalize preceding experiments we explore the effect of large scaling factors using our two image "ensembles". The scaling behavior of histograms of oriented gradients over a scale range is displayed in Figure 6.2. The linear increase of the average with increasing scaling factor again is in agreement with Equation (6.8). Thus we can estimate the gradient histogram values over image regions for multiple scales by applying the multiplicative constant roughly to the scale change. Analyzing the figure on the right shows the hypothesized result of the estimation becoming less accurate for larger scale changes. As expected the standard deviation increases proportional to the scale difference.

The results obtained in this section show that it is possible to approximate gradient histograms in both upsampled and downsampled images using gradient computed merely at the original scale. Although gradient histograms within individual images can change, the gradient histograms over a set of images will be related roughly by a multiplicative constant. For upsampling this constant will be roughly equal to the scale change between the images. The case of downsampling differs because image structure is lost, hence the multiplicative constant is not the inverse of the scaling factor but somewhat less. The agreement between predictions and observations is accurate for typical images but can deviate for individual images, in particular those with a significant amount of high-frequency content or with little to no structure, *i.e.* low average gradient magnitude.



(a) Car images ($k = 2$)

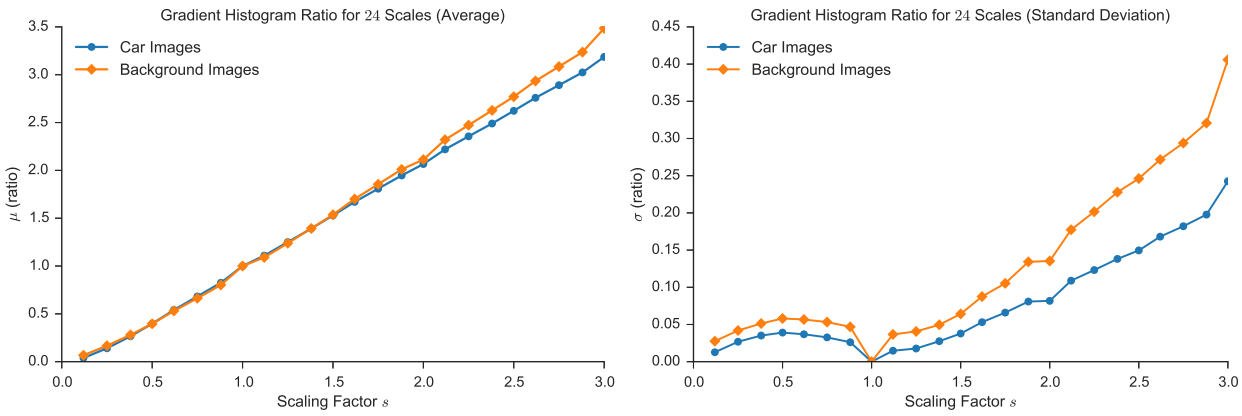
(b) Background images ($k = 2$)

(c) Car images ($k = 0.5$)

(d) Background images ($k = 0.5$)

Figure 6.1: Relationship between gradient orientation histograms measured as ratio r_q for upsampling and downsampling using bilinear interpolation. We show the result for one orientation bin. Results for other bins are similar and are omitted. For upsampling, the mean $\mu \approx 2$, as expected from relationship (6.8). The downsampling case is more interesting, and we find $\mu \approx 0.33$. This is also in agreement with theoretical hypothesis given in (6.9).

The analytical expressions derived for scaling the contents of histogram bins for oriented gradients are an exception. For different types of features it is very hard if not impossible [15] to derive such relationships. In the following sections we will no longer focus on histograms of oriented gradients but rather discuss a general framework for estimating image statistics over scales. We will rely more on empirical observations since an analytical approach is no longer feasible. Fortunately the image processing community has devoted large interest to image statistics, primarily with applications in television, video compression, texture analysis and denoising [81]. In the next section we discuss interesting results regarding the scaling of image statistics and their application to feature approximations for object detection.



(a) Average **(b)** Standard Deviation

Figure 6.2: Scale relation for gradient orientation histograms over 24 scales. We display the average ratio μ and the standard deviation σ for one gradient orientation channel (other channels are similar). Note how the standard deviation drops to zero at $s = 1$ since this corresponds to the original scale.

6.3 Scale Invariance of General Image Statistics

In the period 1980–1990 there seems to have been an considerable interest for both image statistics [70, 85, 26] and scale space theory [50, 44, 29]. Both distinct research areas have explored the behavior of image statistics with scale changes and the concept of scale invariance. Although these research areas have significant overlap, the absence of cross references between these works hint that the theories are shaped independently of each other. However the literature tends to converge in more recent years, for example in [86, 95]. The empirical studies by Ruderman, Bialek [69] and Field [26] on image statistics in television signals are most useful in relation to our work. These works examine the statistics of natural images and focus on the similarity between the statistics of *power spectra* at different image scales [86]. Results indicate that some image statistics are scale invariant, hinting at interesting applications.

6.3.1 Scale Invariance in Image Ensembles

Given an input image $i(x, y)$ of size $N \times N$, the two-dimensional signal can be decomposed in its frequency components using the discrete Fourier transform [59]:

$$I(f_x, f_y) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} i(x, y)\, e^{-j2\pi(f_x x + f_y y)} \tag{6.10}$$

$$= A(f_x, f_y)\, e^{j \cdot \Phi(f_x, f_y)}. \tag{6.11}$$

Where f_x and f_y are the discrete spatial frequencies. The power spectrum is then given by its amplitude $A(f_x, f_y) = \left|I(f_x, f_y)\right|$ and phase $\Phi(f_x, f_y)$. Statistics of natural images have found to follow particular regularities: more specifically, natural image spectra follow

a *power law*. This property was first reported in a 1957 paper by Deriugin [12], who performed experiments on television signals and natural images. This work and later also the works by Tolhurst *et al.* [85], Srivastava *et al.* [81] and Zoran *et al.* [95] confirmed these observations. The model of the *mean power spectra* (using polar coordinates) for natural images can be written as

$$A(f) \approx \frac{A_s(\theta)}{f^{2-\eta(\theta)}} \tag{6.12}$$

in which the shape of the spectra is a function of orientation [86]. The function $A_s(\theta)$ is an amplitude scaling factor for each orientation and $2 - \eta(\theta)$ is the frequency exponent as function of the orientation. For natural images it is observed that η clusters around zero [69]. Related studies cited above found a bias in the distributions of orientations, *i.e.* in real world images, including both natural landscapes and man-made environments, horizontal and vertical orientations are more frequent than obliques [1, 86, 81]. In words, the power spectrum is a function of a single angular scale given by the spatial frequency, and it changes as a power of that scale. This means that doubling the spatial frequency always reduces the power of the signal by a factor of $2^{-2+\eta}$. This is *not* true, for instance, for signals with a power spectrum of the form $A(f) \propto e^{-f/f_0}$, where f_0 acts as a "intrinsic" spatial frequency. In addition to this property, natural images are also extremely non-gaussian, *i.e.* the distribution of image statistics possibly have large peaks and heavy tails. Zoran *et al.* [95] present an excellent overview of kurtosis in image statistics distributions.

Starting from the analysis of power spectra, various important empirical observations on scale invariance in natural images have been made. When considering all possible directions of camera or eye, statistics of natural images are expected to be scale invariant [27, 69, 85]. In other words, the marginal distribution of statistics in images remains unchanged when images are scaled. This effect is explained by the fact that ensembles of signals obtained after rescaling is *self-affine* to the original ensemble of signals. This means that the new set of signals must be multiplied by a suitable constant after rescaling to make the statistics approximately identical to the original ones. Let $\phi(x)$ now represent a two-dimensional signal and $Q[\phi(x)]$ denote some ensemble statistic that we are interested in. For natural images, according to Ruderman and Bialek [69] the ensemble statistic of this signal at scale s can be approximated from the statistic at the original scale by the following relationship governing the scale invariance relation:

$$Q\left[\phi(x)\right] = Q\left[s^{\lambda_\phi}\,\phi(sx)\right] \tag{6.13}$$

Where λ is a universal exponent related to the image statistic, independent of both s and Q. Therefore, Ruderman and Bialek argue that for a scale invariant image ensemble we can make the replacement $\phi(x) \to s^{\lambda_\phi}\,\phi(sx)$ for any ϕ representing natural images and expectation value. The authors confirm this relationship empirically by collecting natural image data and performing an in-depth analysis of the image statistics at various scales [71, 68, 70]. Based on this empirical study the authors confirm the scaling hypothesis given in (6.13). This is a strong statement, since it greatly restricts the form of the image distribution. Also it is a property which gives us some intuition about natural images, for

instance it reinforces the notion that objects can appear at any angular scale in an image, *i.e.* they can be any distance away.

In the experiments performed by Ruderman and Bialek [70], two image ensembles I_1 and I_2 consisting of the same images at scales s_1 and s_2 are studied. Given an arbitrary (scalar) image statistic $\phi(I)$ and its expectation $E[\cdot]$ over an ensemble of natural images, they discovered empirically that the relationship between the two expectation values $E[\phi(I_{s_1})]$ and $E[\phi(I_{s_2})]$ of any image statistic depends only on the relative scale s_1/s_2. More specifically the expectation values of the image statistics are related to each other by the power law for image scaling equivalent to the previous introduced relationship after rearranging:

$$\frac{E[\phi(I_{s_1})]}{E[\phi(I_{s_2})]} = \left(\frac{s_1}{s_2}\right)^{-\lambda_\phi} \tag{6.14}$$

Where coefficient λ_ϕ is specific to the corresponding image statistic ϕ. The mathematical derivation of this relationship is given in Appendix A. This powerful result shows that the marginal distributions of statistics of natural images remains unchanged if the images are scaled. The power law is a manifestation of the scale invariant nature of images [81]. It must be emphasized that Ruderman and Bialek only studied this relationship for large image ensembles; statistics of individual images vary across scales. However in the context of our work we are interested in to what extend this powerful result holds for *individual* images. More recently, building upon this work, Dollár *et al.* [13] have shown that the power law also holds approximately for individual images by decomposing an image into a set of smaller image patches, forming a small ensemble. In the next section we will summarize this image decomposition in the context of channel features.

6.4 Power Law for Feature Scaling

In the previous section we have introduced the power law for image scaling, a compelling result with interesting applications. Recently Dollár *et al.* [15, 13] proposed the idea of exploiting the power law for multiscale analysis in an object detection framework. The power law is expected to hold for any image statistic, hence it should also hold for pixel averages or summations over image channels. In other words, the power law can be utilized for approximating channel features over scale ranges. Given this interesting application we replace the general image statistic $\phi(I)$ with channel feature $f_\Omega(I)$ in Equation (6.14). Consequently we introduce λ_Ω as constant in the exponent to denote the correspondence to the channel. We also make a slight modification to our definition of channel features introduced Section 4.1 by including the width and height of the features explicitly. In other words channel features are then redefined as a global mean of C_s over the locations i, j inside the window:

$$f_\Omega(I_s) \equiv \frac{1}{h_s w_s} \sum_{i,j} C_s(i,j) \qquad \text{where } C_s = \Omega(I_s). \tag{6.15}$$

Although this definition is given as global mean, the following discussion also holds for local means (*e.g.* local histograms). This redefinition is given for sake of simplicity in introduction of the power law for feature scaling, note that the feature type is not actually changed but only the dimensions of the rectangular window are included that expresses the feature as average over the window rather than the sum.

Original studies on scale invariance of natural images all focus on statistics computed over ensembles of images. However, in the context of our work we are interested in the scale relationship for individual images. The power law (6.14) holds over entire image ensembles. Fortunately in the recent work by Dollár *et al.* [13], it was shown that an individual image can be decomposed into an image ensemble by breaking it up in a set of patches. Since our definition of channel features Ω is required to be shift invariant, we can explain $f_\Omega(I)$ defined in (6.15) as computing the average of $f_\Omega(I^k)$ over every patch I^k of image I. In other words we can decompose an input image I into K smaller image patches $I^1 \dots I^K$ such that $I = [I^1 \dots I^K]$. Since we have restricted our channel transformations Ω to be shift invariant, the following holds ignoring boundary effects:

$$\Omega(I) = \Omega\left(\left[I^1 \dots I^K\right]\right) \approx \left[\Omega(I^1) \dots \Omega(I^K)\right] \tag{6.16}$$

Substituting these transformations for each patch into the definition (6.15) of channel features and averaging over the patches results in $f_\Omega(I) \approx \frac{1}{K} \sum f_\Omega(I^k)$ over all patches k. However since $I^1, \dots, I^K$ can be considered as small image ensemble this average is also equal to:

$$f_\Omega(I) \approx E\left[f_\Omega(I^1), \dots, f_\Omega(I^K)\right] \tag{6.17}$$

Therefore the power law is also predicted to hold for single images, since by substituting $f_\Omega(I_{s_1}) \approx E[f_\Omega(I^k_{s_1})]$ and $f_\Omega(I_{s_2}) \approx E[f_\Omega(I^k_{s_2})]$ over all patches k into (6.14) we obtain the **Power Law for Feature Scaling**.

$$\boxed{\frac{f_\Omega(I_{s_1})}{f_\Omega(I_{s_2})} = \left(\frac{s_1}{s_2}\right)^{-\lambda_\Omega} + \mathcal{E}} \tag{6.18}$$

Where $\mathcal{E}$ indicates the deviation from the power law and λ_Ω is the power law constant characteristic for a certain channel Ω. We emphasize that this power law should be used with caution, we cannot expect the relationship to hold for small feature regions. In the next sections we perform a series of experiments to obtain better insight into what features will result accurate approximations over image scales. But before doing so we first determine the parameter λ_Ω for our choices of Ω, *i.e.* our channel types. In agreement with [13] we verify that Equation (6.18) is a surprisingly good fit for multiple channel types and image sets (results in Chapter 7). In other words, these experiments show that we can fit λ_Ω such that $E[\mathcal{E}] \approx 0$.

6.4.1 Empirical Estimation of Channel Coefficients λ_Ω

In this section our goal is to empirically validate the power law for feature scaling and determine channel characteristic λ_Ω values. Considering that the power law for feature scaling holds for our channels, we need to determine λ_Ω for each channel to approximate features in a detection framework. In this section we discuss the process of estimating λ_Ω for a certain image channel Ω. To determine λ_Ω for all channels we first choose two image ensembles: our TomTom dataset of rear-view car images and a set of background samples selected at random from the full TomTom panoramas. Like before we treat downscaling and upscaling separately and rescale the original images to 24 scales in both directions, $s = 2^{\pm 1/8}, \ldots, 2^{\pm 24/8}$ where the sign indicates the scaling direction. We calculate the average ratio between feature responses for all 24 scale measurements and all channels. The average ratio of computed as follows, where the summation is computed over all $N = 2000$ images separately for positive and background samples.

$$\mu_s = \frac{1}{N} \sum_{i}^{N} f_\Omega(I_s^i) / f_\Omega(I_1^i). \tag{6.19}$$

Assuming the power law (6.18) holds, we expect the average to be a function of the relative scale difference in the following relationship $\mu_s = s^{-\lambda_\Omega} + E[\mathcal{E}]$. Our goal is to empirically validate this relationship, determine the constant λ_Ω and show $E[\mathcal{E}] \approx 0$ over an ensemble of images. As before rescaling is performed using bilinear interpolation. Since image dimensions are rounded to the nearest integer, scale values are computed as $s' = \sqrt{h_s w_s / hw}$ where h and w are the dimensions of the (rescaled) images. It was observed in [13] that for practical use the power law should be slightly modified due to interpolation artifacts. More specifically, the ratio μ_s does not start exactly at 1 as one would expect. This minor offset difference is explained by interpolation artifacts introduced by blurring even by very small scale changes. Therefore, the power law is modified accordingly by introducing factor a_Ω related to this offset:

$$\mu_s = a_\Omega \cdot s^{-\lambda_\Omega} + E[\mathcal{E}] \tag{6.20}$$

where s now denotes the relative scale difference and $a_\Omega \neq 1$ adjusting for this small offset introduced by interpolation artifacts. This relationship between relative scale and average ratio μ_s is used for finding the optimal values for λ_Ω and a_Ω for each of the 10 image channels. Finding the best values is performed by least squares fitting, of (6.20) after taking the logarithm of both sides and rearranging as follows.

$$\begin{aligned} \log_2(\mu_s) &= \log_2(a_\Omega \cdot s^{-\lambda_\Omega}) \\ &= \log_2(a_\Omega) + \log_2(s^{-\lambda_\Omega}) \\ &= \log_2(a_\Omega) - \lambda_\Omega \cdot \log_2(s) \\ &= a'_\Omega - \lambda_\Omega \cdot \log_2(s). \end{aligned} \tag{6.21}$$

This allows us to simply fit $y = ax + b$ and set $a_\Omega = 2^{a'_\Omega}$.

The least squares fitting results are displayed in Figure 6.3 and Figure 6.4 for downscaling and upscaling respectively. From these plots we observe that μ_s follows a power law for all channels as hypothesized. In other words (6.20) is an excellent fit for the observations on the average ratio between feature values at the different scales μ_s for both image ensembles. However comparing our plots to results reported in [13] we conclude that the curves for our image ensembles are more apart hinting at different λ_Ω values for both ensembles. This poses a serious problem since in an object detection setting we obviously do not know whether we are dealing with object or background patches hence choosing λ_Ω impossible. Therefore we decide to perform least square fitting over the combined set of car and background ensembles and adopt the combined λ_Ω values.

From our analysis of Figure 6.3 we conclude that for downscaling with $s < 0.4$ it seems unreasonable to expect accurate approximations since the curves for both ensembles start to diverge significantly. This divergence is partially explained by interpolation artifacts and removal of high-frequency content in the downsampling process, although less dramatic effects are reported in [13]. Better explanation to the difference in results could be due to the fact that Dollár *et al.* exclude nearly uniform image windows (*i.e.* with low average gradient magnitude) from the background image set. Close inspection of the images comprised in our background set indeed shows a considerable amount of image patches with almost no image structure, *e.g.* road or sky. However we would argue that removing background samples with small average gradient magnitude is flawed since feature approximations are also calculated for structureless image patches when applied in an object detector. Hence the λ_Ω values should be determined based on image patches as much as possible representing real world settings.

Similar plots for upscaling in Figure 6.4 show that curves for both ensembles are closer to each other. In comparison to the downscaling plots this can easily be explained by the fact that no image structure is removed in the upscaling process. Pixels are simply upscaled with bilinear interpolation resulting in smoother curves in comparison to downscaling. Also note that in both figures the color channels show an horizontal line indicating $\lambda_\Omega = 0$ as expected.

These plots clearly show the power law behavior of the image statistics for our 10 image channels. To demonstrate the validity of the power law and its robustness, we determine the scaling parameters independently over two different image sets captured with different cameras, featuring non similar image statistics. This time we sample uniformly at random 50×50 image patches from the TME dataset and the TomTom panoramas. Using this approach we ignore the distinction between car and background samples and set a smaller image size for a more realistic setting to our actual feature windows which are typically smaller. We argue that this method of feature sampling is a better reflection of real world scenarios since it captures a wider variety of image windows. In comparison to the previous plots we also limit our scale range, as in practice we will only use approximations over a single octave. To obtain more precise fits we compute the best linear least squares fits over the range $s = \left[\frac{1}{2}, 1\right]$ and $s = [1, 2]$ treating cases of upscaling and downscaling separately. Resulting plots for both image datasets are displayed in Figure 6.5.

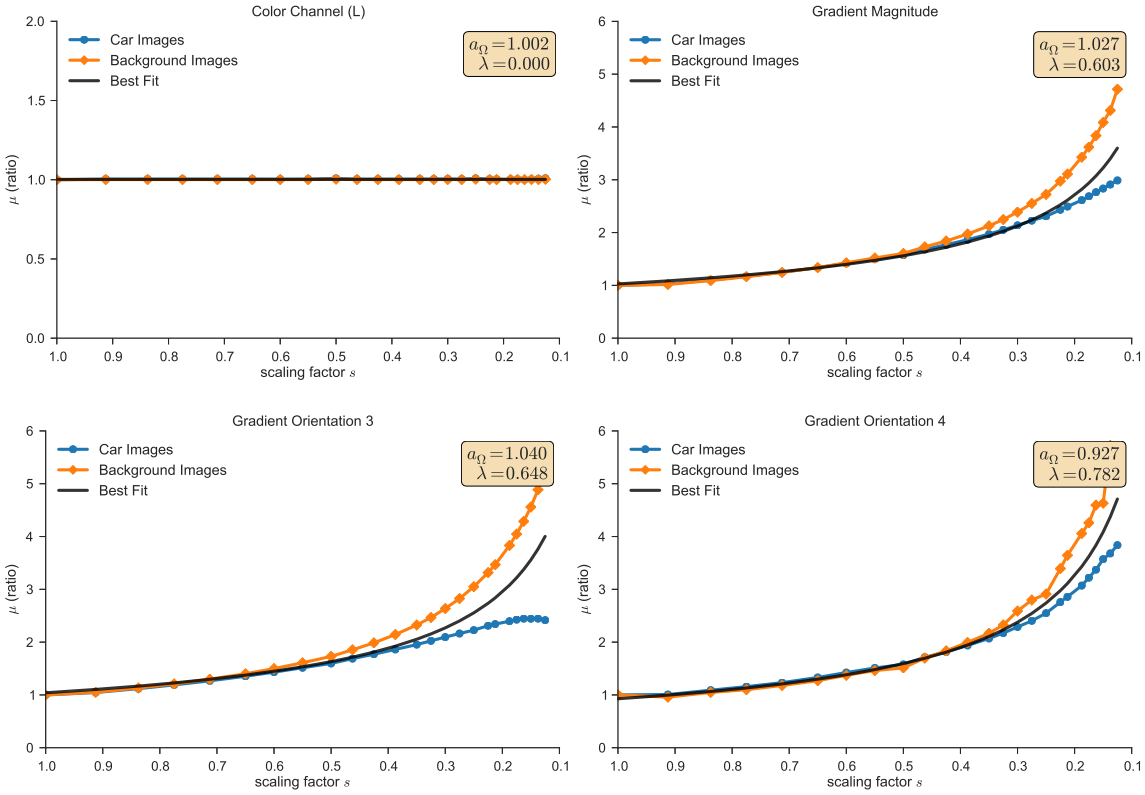


Figure 6.3: Calculation of factors λ_Ω and a_Ω for **downscaling**. Feature ratios are computed over 2000 car images and 2000 random background patches of size 80×80 from the TomTom dataset. Least squares fitting results for (6.21) are indicated in the top right corner.

Minor differences aside we observe that the figures are nearly identical for both image sets. This confirms our previous findings and displays the general robustness of the power law for images captured with different cameras. Although the exact same channel computations are applied to both image sets, we make the interesting observation that for downscaling some of the curves seem to be swapped. Comparing Figure 6.5 (a) and (c) we observe the gradient orientation channels 1 and 3 to be interchanged. Given that the corresponding orientations of these channels are not adjacent (see Figure 4.2), we find it hard to explain this behavior. However this observation together with the fact that the rest of the gradient magnitude and orientation channels are almost identical, exhibits that we can use the same λ_Ω parameter for all gradient channels. The final parameters used in our object detector for performing multiscale approximation are the fitting results of Figure 6.5 over the TomTom dataset. In Chapter 7 we perform a series of experiments to test the approximation accuracy of the power law for channel features.

6.5 Feature Approximations for Object Detection

In the previous sections was have introduced the power law for feature scaling and described the method for experimentally determining the channel characteristic scaling

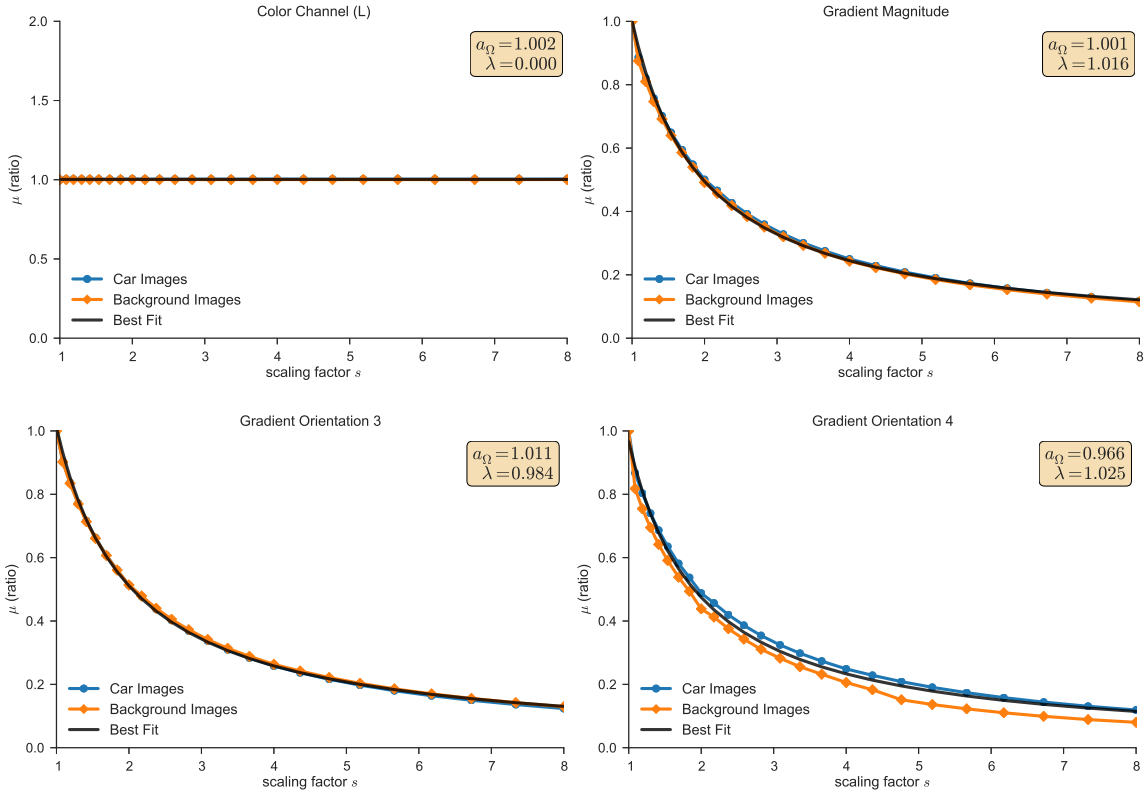
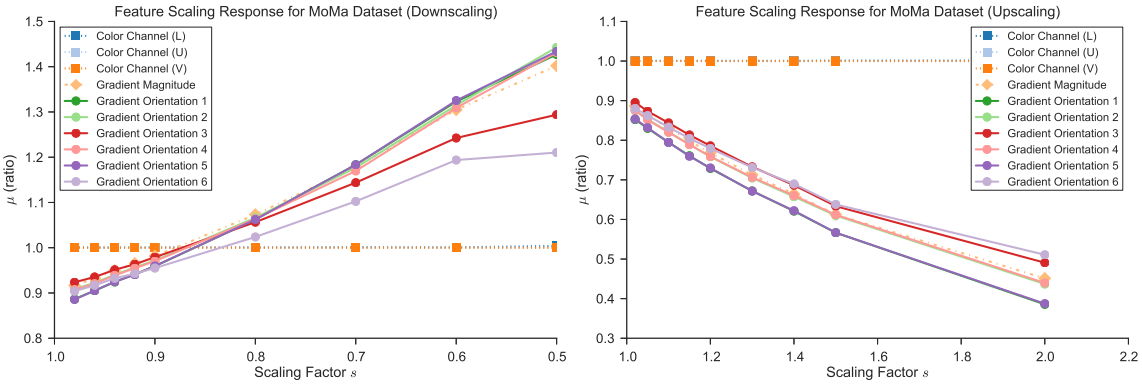


Figure 6.4: Calculation of factors λ_Ω and a_Ω for **upscaling**. Feature ratios are computed over 2000 car images and 2000 random background patches of size 80×80 from the TomTom dataset. Least squares fitting results for (6.21) are indicated in the top right corner.

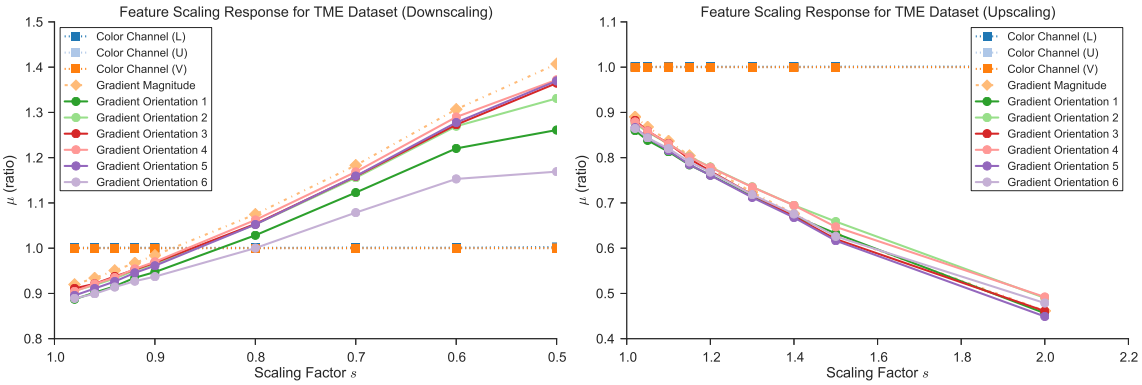
factor λ_Ω for each channel. The power law for feature scaling given in Equation (6.18) allows for estimating channel features at different image scales than the channels are available at. This is an powerful result that opens up doors for performing fast object detection with computing the image pyramid for a dense number of scales. This idea was first introduced by Dollár *et al.* [15] and later extended by Beneson *et al.* [3] to built extremely fast object detectors. In this section we place the feature approximation technique in context of our object detection framework. For this we assume that feature approximations are accurate enough, although experimental results to confirm this are discussed later in Chapter 7.

The power law governing feature scaling can directly be extended to an application using channel images. Let I be an image and let I_s capture the same content captured at scale s. We denote image I resampled by a factor s as $R(I, s)$ closely following the notation in [13]. In the context of object detection using channel features we typically compute the channel image $C = \Omega[I]$ at the original scale. For computing the channel image at a different scale s the standard approach is to calculate $R(I, s)$ and recompute the channel, *i.e.* $C_s = \Omega[R(I, s)]$. This ignores the information obtained from computing $C = \Omega[I]$.



(a) TomTom dataset – Downscaling

(b) TomTom dataset – Upscaling



(c) TME dataset – Downscaling

(d) TME dataset – Upscaling

Figure 6.5: Scaling factors determined by alternative method on two different datasets. Averages μ_s are computed over $N = 20000$ randomly sampled image patches of 50×50 pixels.

Dollàr *et al.* [13] were the first to propose computing the channel at scale s directly from the channel at the original scale using the power law approximation:

$$C_s \approx R(C, s) \cdot s^{-\lambda_\Omega} \tag{6.22}$$

This approximation directly follows from the power law (6.18) by substituting $s_1 = s$ and $s_2 = 1$ and rearranging. Equation (6.22) is an important result that demonstrates that we can approximate the *feature pyramid* required for multiscale object detection.

6.5.1 Fast Feature Pyramids

A *feature pyramid* is a multiscale representation of an image I, where channels $C_s = \Omega[I_s]$ are computed for a large number of scales s. Scales are typically sampled evenly in log-scale, starting at $s = 1$, with typically 4 to 8 scales per octave [13]. Let N denote the total number of scales for which we want to detection objects. As we have seen throughout this work, the standard approach for constructing the feature pyramid is to compute

$C_s = \Omega[R(I, s)]$ for every scale s. In the context of our work this means that we compute the image pyramid and extract channels for each level. This standard approach in object detection was illustrated in Figure 4.7 for the introduction of our baseline classifier.

The channel approximation (6.22) suggests a straightforward method for efficient computation of the feature pyramid without rescaling the input image several times. Dollár *et al.* [13] propose a hybrid method for constructing the feature pyramid. First the channels $C_{s'} = \Omega[R(I, s')]$ are computed at just one scale per octave: $s' \in \left\{1, \frac{1}{2}, \frac{1}{4}, \ldots\right\}$. Let K denote the number of scales s'. From these so-called *base scales* the channel approximation is used for computing the channels at intermediate scales s:

$$C_{s'} \approx R\left(C_{s'}, s/s'\right)\left(s/s'\right)^{-\lambda_\Omega} \tag{6.23}$$

Where s' represents the nearest base scale available. Thus the intermediate scales can be approximated from the base scales, for which the channel images are explicitly computed. Computation of the channels for the base scales $C_s = \Omega(R(I, s))$ at one scale per octave is reported [13] as good trade off between speed and accuracy. Increasing the number of base scales obviously increases computation time significantly. However adding more base scales also decreases the maximal scale distance for which approximations are computed, hence the approximation will become more accurate. Note that in this proposed method, the image is rescaled N/K times.

Figure 6.6 displays the proposed hybrid approach for computing the dense feature pyramid using the channel approximation in (6.22). For our framework using integral channel features an alternative application of the power law governing feature scaling is suggested [13]. Instead of rescaling the channels as as discussed above, one can instead rescale the *detector*. The power law (6.18) can be applied to approximate features at nearby scales and use the corresponding feature responses for detecting objects at these scales. As a result we only need to compute the integral channels for the base scales and rescale our detector to perform object detection over all intermediate scales. This idea was further extended by Beneson *et al.* [3] who propose a framework for object detection without image rescaling.

6.5.2 Object Detection without Image Rescaling

In the hybrid approach of Dollár *et al.* [13] described in the previous section, the input image is rescaled N/K times and the intermediate $N - N/K$ scales are processed using approximations by the power law. Not surprisingly this significantly reduces the detection time, *i.e.* the authors report a 2.5× performance increase by implementing this method. Beneson *et al.* [3] built upon the multiscale feature approximation techniques by proposing object detection without image resizing. Their core idea is to move resizing of the image from detection time to training time. The authors propose training *multiple* detectors, one for each base scale. The detectors trained at base scales are then used as starting point for performing multiscale detections at all intermediate scales by computing the approximated feature responses using the power law. We now describe how in the context

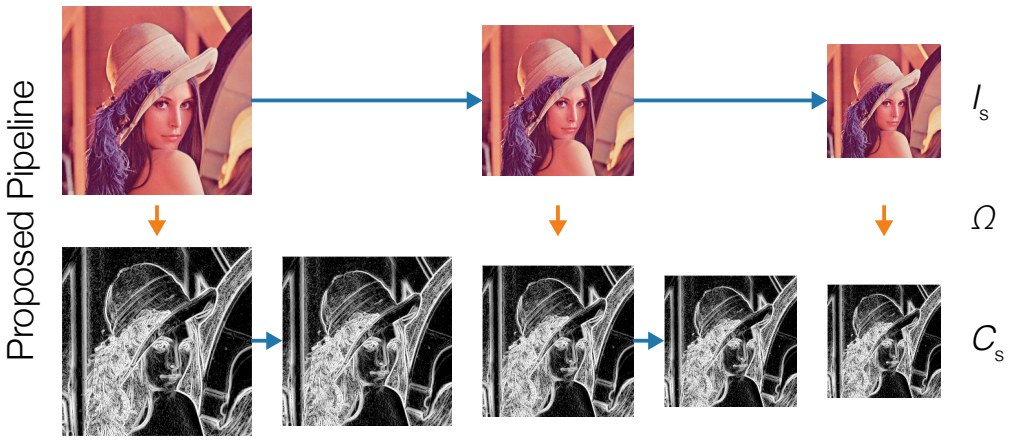


Figure 6.6: Proposed detection [13] pipeline with hybrid feature pyramids. Channels C_s are approximated from base scales using the power law. Blue arrows in the bottom row indicate approximations which are computationally inexpensive compared to recomputing channels explicitly for that scale.

of our detection framework using channel features and boosting we can perform object detection without image rescaling.

First recall, that in our boosting framework a sequence of decision stumps $h_t(x)$ each corresponding to a single channel feature is used for classification of a subwindow. Since the power law allows us to approximate feature responses across scales, we can decide how to adjust a given stump classifier in order to classify an object for which the features are approximated. Each stump in our classifier is described by a decision threshold τ and a single channel feature characterized by a channel index and rectangular region (see Section 4.2 for details). For rescaling a trained detector with factor s, all decision stumps and feature rectangles must be rescaled. More precisely, the channel index is kept constant, the rectangle is scaled with factor s and the decision stump threshold τ is changed according to the power law. Let $r(s)$ denote the ratio between a feature response at scale 1 versus scale s, then according to the Equation (6.20) this factor can be written as:

$$r(s) = \begin{cases} a_u \cdot s^{-\lambda_u} & \text{if } s \geq 1 \\ a_d \cdot s^{-\lambda_d} & \text{if } s < 1 \end{cases}. \tag{6.24}$$

Where the subscripts now explicitly indicate the different coefficients used for upscaling and downscaling. The values for the channels in out implementations are determined by the experiments from Section 6.4.1. Our channel features detector requires only slight changes for fast multiscale detections using feature approximations. When rescaling a stump by a relative scale factor s, we keep the channel index constant, scale the rectangle by s and update the threshold as:

$$\tau' = \tau \cdot r(s). \tag{6.25}$$

The scaling factor $r(s)$ corresponds to the channel index of the feature for that decision stump. The modification for decision stumps and the detectors trained at multiple scales allow for object detection without image rescaling. By taking a detector trained at a base scale, we can convert it into K classifiers for slightly different scales using the modification of decision stump thresholds. Based on this approach, a number of N/K classifiers is trained, one for each image octave: $s \in \left\{1, \frac{1}{2}, \frac{1}{4}, \ldots\right\}$. While for many object detectors this is not practically feasible due to excessively long training times, our WaldBooost detector trains in approximately 40 minutes. Consequently training the detector at four different scales can be done in 2.5 hours on a fast desktop using a parallel implementation. At runtime we use the four models at different scales as base scales and apply the power law for feature scaling (6.18) to approximate intermediate scales.

At detection time, the N/K classifiers are converted into N classifiers (one per scale). Instead of computing the integral channels for each scale, this approach only requires the integral channel for one scale. To perform multiscale detections the N classifiers are applied using the traditional sliding-window technique. As we will see in Chapter 7, eradicating the need for image rescaling and channel computation at runtime, significantly increases detection speed, opening the door for real time object detection. However we should not forget that the method relies on the power law for feature scaling and that the approximations are guaranteed to have a negative effect on the detection quality.

6.6 Chapter Summary

In this chapter we focused on improving multiscale handling of our object detector. After a lengthy discussion on the theory of image statistics under scale changes we gradually worked towards building a multiscale object detector without image rescaling. The foundation for this idea is the power law for image statistics, which describes the powerful finding that the ratio in the expectation value of arbitrary image statistics captured at two different scales is only a function of the relative scale difference. Although the power law holds in general for various image statistics, each statistic has its own constant λ that can be determined empirically. For integral channel features, it was shown that the power law can be used for constructing a feature pyramid without explicit computation of the image pyramid. This idea was then extended by moving the image rescaling from detection time to training time, *i.e.* training multiple detectors for a number of base scales. In conclusion, the core insight of the theory described in this chapter is that feature responses of nearby scales can be approximated with an important application for object detection.

7

Detection Framework and Experiments

In this chapter we bring together all the parts discussed in the previous chapters. With our baseline detector as strong foundation we improve it by adding WaldBoost and feature approximations for better multiscale handling. These components describe our overall detection framework. The first two sections of this chapter discuss the overall *learning* and *detection* framework consisting of the following components:

- **Baseline detector** with integral channel features and Discrete AdaBoost.
- **WaldBoost** for soft-cascade detection.
- **Multiscale approximations** using the power law for feature scaling.

After discussing our overall detection framework, the remaining part of this chapter discusses a series of experiments both on the complete object detector and the individual components. Our first experiment illustrated the feature representation for rear-view car detection using integral channel features, then we provide insight in WaldBoost as classification algorithm and perform a large number of experiments on the feature approximation accuracy introduced in Chapter 6. In the section thereafter we perform two small experiments on determining the robustness to translation and scaling of our baseline classifier. Finally in the last sections of this chapter we report precision-recall curves indicating the detection quality of our detector and speed benchmarks. Throughout this chapter we also present a number of visualizations and videos for illustrating the workings of our detector.

7.1 Learning Framework

The overall learning framework is almost identical to WaldBoost training process described in Algorithm 3. In this section we describe the procedure in context of integral channel

features and explain how we train multiple detectors over a number of base scales as discussed in Section 6.5.2 – Training is initialized by specification of the parameters summarized in Table 7.1. The possibility exists to train a detector using Discrete AdaBoost; this renders the WaldBoost parameters unnecessary. In the following discussion we focus on WaldBoost training since this procedure is used in our final detection framework. As explained in Section 6.5, our detector is trained at four different scales to enable fast multiscale detections without image rescaling in the detection phase. The model width and model height in the table of parameters correspond to the base scale $s = 1$. The size of the three remaining base scales is computed from the model size at the original scale. For training our car detector with a base model size of 64×64 this corresponds to training the detectors at sizes 32×32, 64×64, 128×128 and 256×256. Training is performed for each scale independently, starting with the smallest scale working upwards.

Training the detector for a given scale begins with loading both training and validation images at the particular model size into memory. This is followed by generating a large candidate pool of channel features constrained by the model size. In other words, the random rectangular regions for each channel feature should be "inside" of the model. After generation of the candidate feature pool, the corresponding feature responses for all training and validation samples are computed and stored in a large data matrix. This process involves extracting the 10 image channels for all training examples, computation of the integral images and evaluating each channel feature. Fortunately this process is executed very rapidly due to the GPU implementation for extracting the channel images. Computing the feature matrix of 40 000 candidate channel features for 4000 training examples takes less than two minutes on `Platform II`.

Once the initialization steps are finished, WaldBoost training takes over by iteratively training weak classifiers until the sufficient number of stages is reached. Details of finding the best decision stump for each training stage are described in Section 4.2.3. Once the optimal weak classifier is determined for a given stage, the classifier updates the sample scores $H_t(x)$ for all training and validation samples. Then non-parametric density estimation (Section 5.4.2.1) for determining the class-conditional probabilities is used after which the stage rejection thresholds are computed using Wald's SPRT. Optionally as last step for each training stage, bootstrapping can be applied for pruning training samples outside of the thresholds and resampling new training data to explore the largest set of training examples. The process of training the strong classifier is performed for all four scales independently. For a strong classifier of 400 weak classifiers, the final multiscale classifier has trained a total of $4 \cdot 400$ weak classifiers each with their own decision threshold, polarity and importance.

The overall training process is relatively fast in comparison to other object detectors such as deformable part-based models or convolutional neural networks. Given a training dataset of 4000 examples and initializing a feature pool containing 40 000 features, the training time of a model at a single scale takes about 40 minutes on `Platform II`. Consequently, training the detector at all four base scales is done in approximately 2.5 hours.

Table 7.1: Training parameters in the overall detection framework. Parameters indicated with superscript † are only required when using WaldBoost.

Parameter	Description	Default
`boost_type`	Boosting type to use (AdaBoost/WaldBoost)	WaldBoost
`weak_count`	Number of weak classifiers to train	600
`feature_count`	Size of the initial feature pool	40000
`model_width`	Width of rigid-template that is trained	64px
`model_height`	Height of rigid-template that is trained	64px
`base_scales`	Base scales (octaves)	0.5, 1, 2, 4
`num_train_pos`	Number of training samples (positive)	2000
`num_train_neg`	Number of training samples (negative)	2000
`min_feature_area`	Minimum feature size	25px
`waldboost_alpha`†	WaldBoost parameter α	0.005
`waldboost_beta`†	WaldBoost parameter β	0
`waldboost_validation_pos`†	Number of validation samples (positive)	500
`waldboost_validation_neg`†	Number of validation samples (negative)	500

For each learning iteration, training the best weak classifier is the most time consuming. This involves seeking the optimal decision threshold for *all* candidate features and then selecting the overall best weak classifier by finding the one with the lowest error over all training examples. In our implementation this process the parallel execution results in a significant speed-up in training time. Our desktop machine runs the process on 8 CPU threads. Additional speed improvement can be achieved by moving the training process to the GPU, however since training is already fast we did not consider this.

7.2 Detection Framework

The overall detection architecture is built around the sliding-window technique that classifies each window independently. The detection phase uses a trained classifier to perform a dense scan to detect objects at all sizes and locations. This process reports *preliminary* object detections for each location of the test image. In the final step of the detection process, nearby preliminary detections are fused to obtain the final detections. For object detection in videos, the overall detection framework remains the same except that decoded video frames are proposed to the detector. Trained detectors, product of the learning stage, are saved and loaded as JSON format[1] and contain the specification of the classifiers for each scale. This is an excessive list of weak classifiers characterized by channel feature, decision threshold, polarity and weight α_t in the strong classifier. After loading a detector from file, parameters summarized in Table 7.2 can be set before performing actual detections.

[1]Example detector file can be found at `https://gist.github.com/tomrunia/9b99341c803cc3dd578c`.

Table 7.2: Detection parameters in the overall detection framework.

Parameter	Description	Default
`boost_type`	Boosting type to use (AdaBoost/WaldBoost)	WaldBoost
`weak_count`	Number of weak classifiers for detection	600
`sliding_window_x`	Sliding-window step (x-direction)	8
`sliding_window_y`	Sliding-window step (y-direction)	8
`scales_per_octave`	Number of intermediate scales	8
`nms_grouping_threshold`	Overlap threshold for NMS	0.5
`boost_final_threshold`	The final boosting threshold	0.5

The architecture of our overall detection framework is displayed in Figure 7.1. Upon receiving a video frame from the camera, the image is sent directly to GPU memory. The GPU executes a set of OpenCL filter kernels for computing the LUV color, gradient magnitude and gradient orientation channels. Next the *integral* images are precomputed for fast feature extraction. Surprisingly enough, our benchmarks show that computing the integral images on the CPU is faster than computing them on GPU. Most likely this is caused by data bandwidth limitations between CPU and GPU. Since the integral channels are large memory objects and sending these back to the CPU introduces some latency. Therefore in our current implementation only the filter kernels are applied on the GPU while computation of the integral images is performed on the CPU. Once computation of the integral channels is finished, classification of all subwindows over all scales is initialized in parallel using all available CPU threads.

We emphasize that *no* image rescaling is performed but rather we use the models trained at four different scales. These base scales are then used as starting point for feature approximations to intermediate scales. In practice, multiscale detection without image rescaling is implemented as follows. For each position of the sliding-window, multiscale analysis is performed at all scales by first finding the nearest scale at which a trained detector is available and then using this detector as starting point for the power law approximation. The relative scale difference between the original scale and the nearest base scale is used for rescaling the channel features and modifying the decision stump threshold according to Equation 6.25. In other words, we *rescale the detector*. After rescaling the feature windows and updating the decision thresholds, the remaining classification steps are performed by WaldBoost. After processing the image in sliding-window fashion and harvesting all preliminary detections, non-maxima suppression is applied for the fusion of overlapping/nearby detections.

7.2.1 Fusion of Multiple Detections

Non-maxima suppression (NMS) is a technique commonly applied as final step in the detection process to fuse nearby preliminary detections that possible correspond to the same object in the test image. In general, less-confident detections are removed while

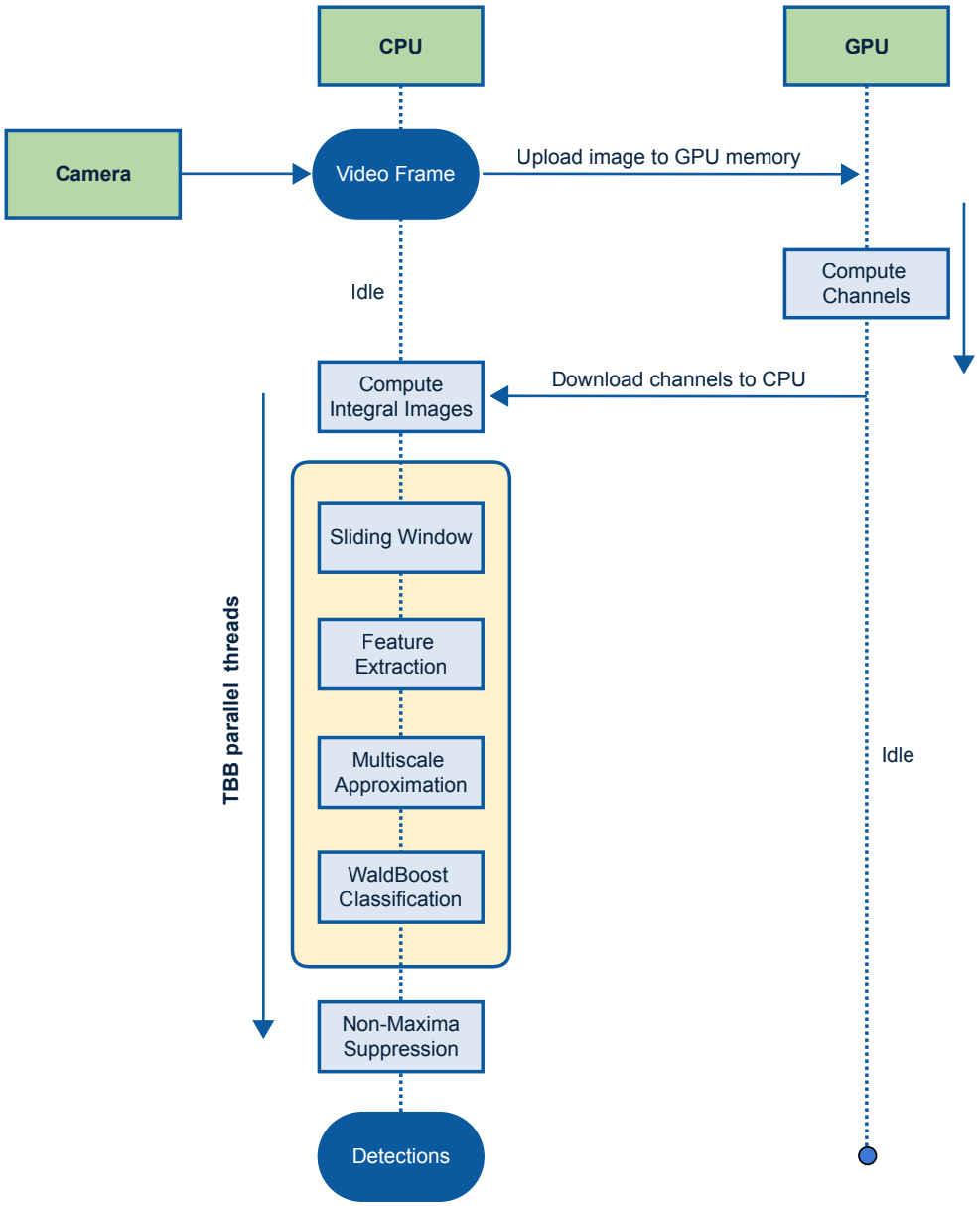


Figure 7.1: ***Architecture of our overall detection framework.*** Upon receiving an image or video frame, the detector uploads the image to GPU memory. The GPU executes the filter kernels for extracting the 10 image channels. After downloading the channels back to host memory the CPU begins classification of subwindows in sliding-window fashion. This process is speed-up using a parallel implementation built on TBB.

detections that have higher scores are kept as final detection. Mean-shift NMS [10] is a popular method, however it has multiple settings that depend on various aspects of the detection such as the step-size. For our work, instead we use a simpler NMS algorithm that uses a straightforward heuristic for determining the final list of detections.

The simple NMS algorithm [16] works as follows. First the algorithm sorts the preliminary detection windows x_i in decreasing order of sample score $H_t(x_i)$. Then starting at the beginning of the list we take the first preliminary detection and compute the PASCAL VOC overlap criterion (Section 3.2.1) for all other detections. If the overlap criterion exceeds the predefined overlap threshold, the detection with the lowest score is removed from the list. Thus, for two overlapping detections we keep the one with the highest score. In the worst case this heuristic takes $O(n^2)$ in the number of preliminary detections n. In practice this procedure is extremely fast since the number of detections typically is less than a dozen per test image. This process typically takes less than 20 microseconds on `Platform I`. For all experiments through this work we set the intersection-over-union grouping threshold to 0.5. With this we conclude the discussion of our overall detection framework. In the remainder of this chapter we perform a series of experiments to study its characteristics, performance and speed.

7.3 Feature Representation using Channel Features

After design and implementation of the overall detection framework, our first interest is the feature representation for rear-view car detection that is learned during the feature selection process. Existing works [4, 16, 3] on integral channel features for *pedestrian* detection show visualizations that give clear understanding of what features are most discriminative for recognizing humans. To our knowledge the problem of car detection has not been approached using channel features, so for that reason we are interested in the feature representation that is learned during the training process. The visualizations and statistics reported in this section are a result of training our baseline detector multiple times and averaging the results. All detectors are trained using our TomTom rear-view car dataset containing 2000 positive and 2000 background examples.

Figure 7.2 displays the spatial distribution of integral channel features per channel for a car detector trained at a model size of 64×64 pixels. Recall that we are using three LUV color channels, one gradient magnitude channel and six gradient orientation channels. For creating this visualization we train our baseline classifier with 400 decision stumps, each corresponding to a single channel feature. In our final classifier, the weight received by each weak classifier is given by α_t which is inversely proportional to the classification error of that stage (Section 4.2). We visualize the total weight $\sum \alpha_t$ per pixel over all channels. This illustrates the spatial distribution of weight in the final classifier. To create this figure, we iterate through all learned features and keep track of the total weight received per pixel within the model size. Blue colors indicate regions that receive almost no weight in the final classifier, while red regions are important in making the final classification. The spatial distribution of features is complemented by the distribution of

weight over the channels given in the bar chart of Figure 7.3. The heatmap and the bar chart together give a clear understanding of what features are most important in the final classification process for rear-view car detection.

Numerous interesting observations can be made from the combination of these figures. The most striking observation is the fact that gradient orientation channel 3 corresponding *horizontal edges* receives significantly more weight than the other channels. More than 25% of the weight in the final classifier is allocated to channel features encoding vertically oriented gradient information[2]. The heatmap for the corresponding channel indicates that feature responses near the roof, license plate and windshield region are most important. To rule out the effects of random feature pool initialization we repeat the experiment a number of times, but results remain similar. Figure 7.4 shows the number of features selected per channel during the training procedure, also indicating the importance of gradient orientation channel 3. As a visual support for the importance of the horizontal edges in the detection process, we display the channel responses for a number of positive and negative training images in Figure 7.5. Looking at these images it is not surprising that a classification between positive and negative samples can be made based on this orientation channels. This distinction is not perceived as strongly for the other channels.

Statistics also show that the gradient magnitude channel contains almost no information for our classification problem, accounting only for 3% of the total weight. The difference in importance between horizontal and vertical edges is also striking. Gradient orientation channels 1 and 5 seem to focus on the shape of the car and are symmetrical to each other. The first color channel has a strong response for the shadow on the ground below the car. In Figure 7.6 we report the aspect ratio and size of channel features that are selected for training a detector at 64×64. From these figures we read that a significant amount of features is either *wide* or *long*.

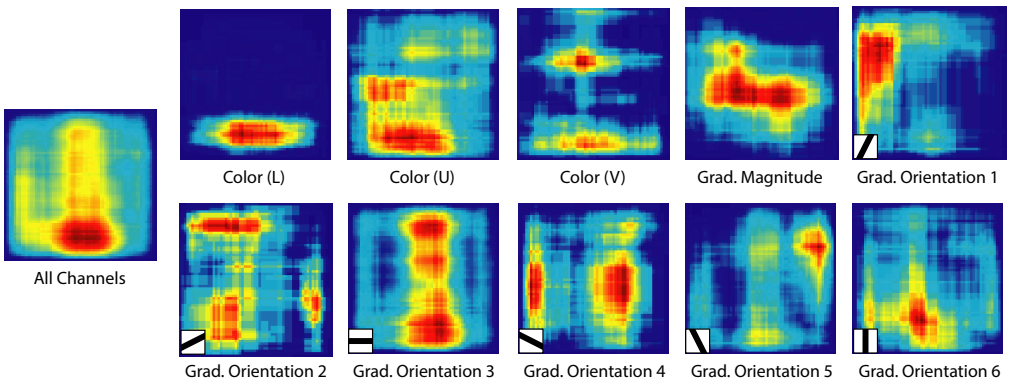


Figure 7.2: ***Spatial Distribution of Features per Channel.*** The heatmap visualizes the total importance in the final classifier for rear-view car detection (64×64 model). Blue encodes regions that receive almost no weight in the final classification while red regions are important for making a decision upon the label.

[2]Horizontal edges correspond to vertically oriented gradients (Gradient Orientation Channel 3).

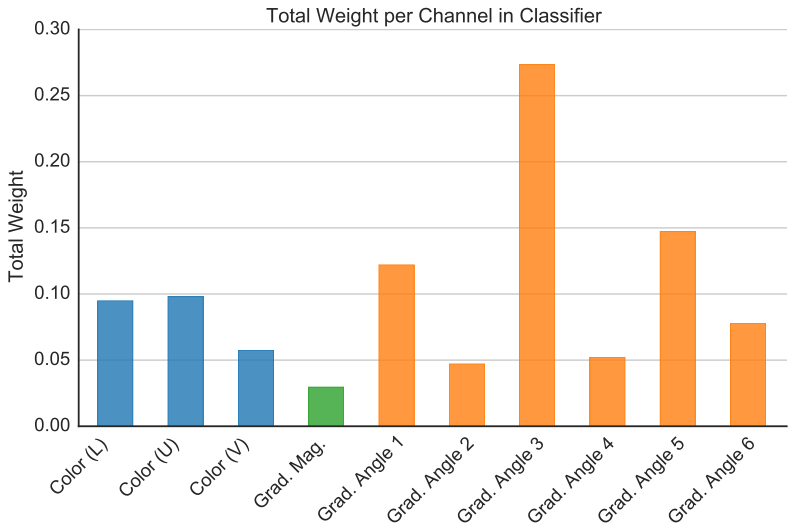


Figure 7.3: ***Total Weight per Channel.*** Distribution of learned channel features over the ten image channels for training on the TomTom rear-view car dataset. The height of the bars indicates the total weight $\sum \alpha_t$ in the strong classifier.

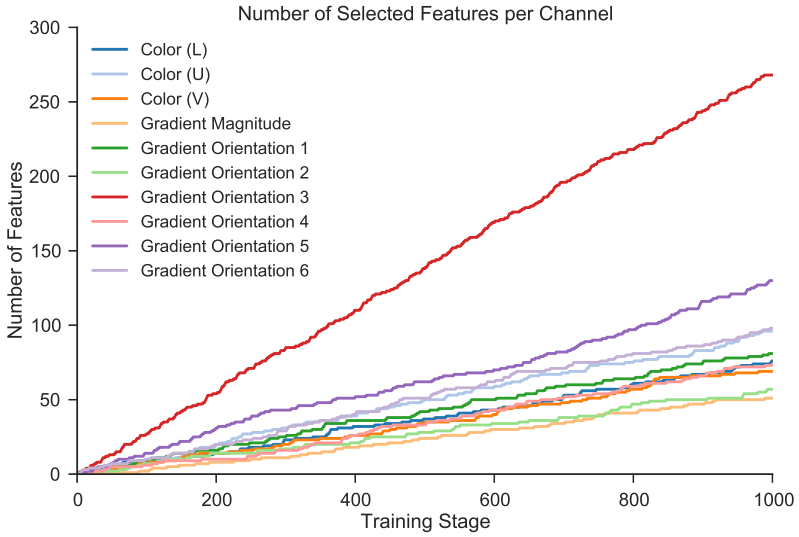
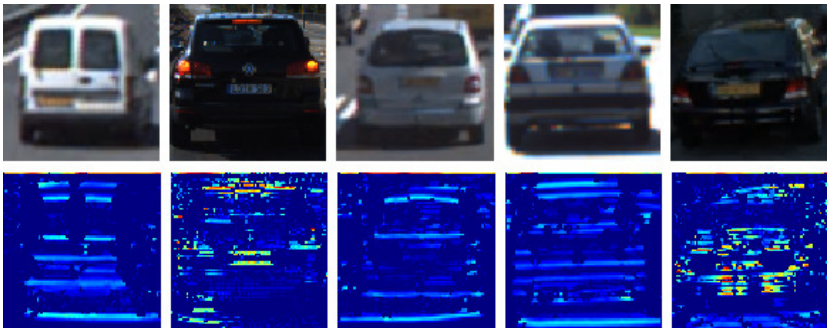


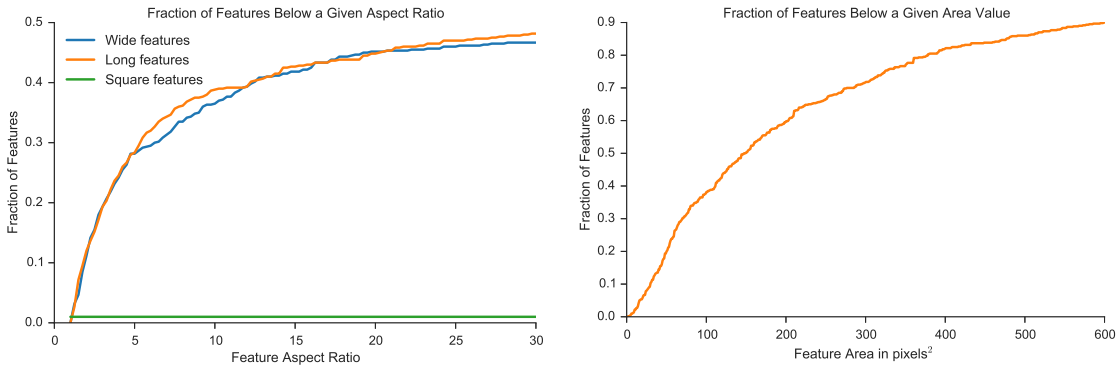
Figure 7.4: ***Number of Features per Channel.*** Cumulative number of features during the training process. Statistics are averaged over multiple training runs each with 1000 weak classifiers. The initial feature pool contains 40 000 channel features.



(a) Positive Samples (cars)

(b) Negative Samples (background)

Figure 7.5: ***Response Gradient Orientation Channel 3.*** Example channel responses for vertically oriented gradients. We observe a clear distinction between the two classes which confirms that horizontal gradients can be used to build a strong descriptor. For the other gradient orientation channels the distinction between positive and negative samples is less noticeable.



(a) Feature Aspect Ratio

(b) Feature Size

Figure 7.6: ***Shape and Size Statistics of Channel Features.*** The cumulative fraction of features below given aspect ratio of feature size are reported for car detection. The fraction of square features is limited, the feature representation is dominated by wide and long features.

7.4 WaldBoost Experiments

The WaldBoost algorithm included in our detection framework is evaluated for the rear-view car detection problem. In this section we present visual proof and understanding of the validity of WaldBoost in the context of our detection framework using channel features. For the experiments in this section the allowed false negative rate α was set to 0.005 while $\beta = 0$. The detector is trained on the collection of 2000 rear-view car examples and 2000 background samples from the TomTom dataset.

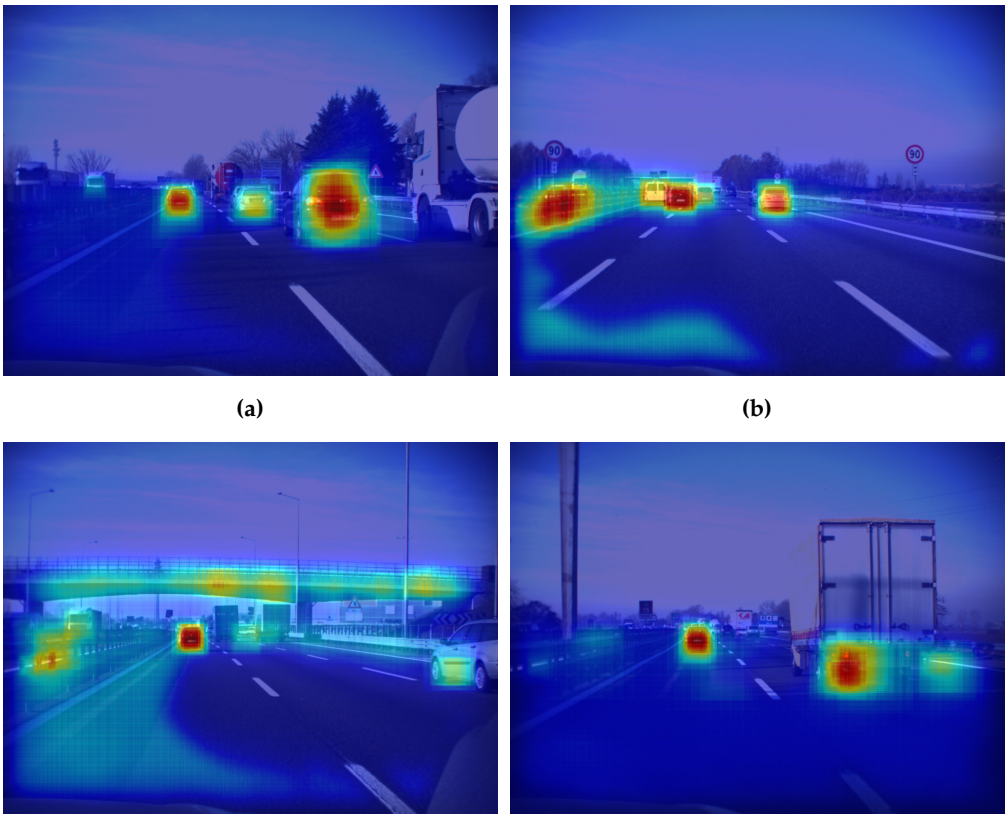
Speed and error rates of the WaldBoost classifier are influenced by the number of weak classifiers used. In our example we train a classifier using $T = 400$ decision stumps. Later in this chapter we report results for our overall detector varying the number of weak classifiers. Note that the application of WaldBoost is neither feature nor weak classifier specific. The average evaluation time to decision $\overline{T}_S$ also referred to as the number of weak classifiers evaluated before rejection is the crucial factor in decreasing the overall time spent on classification of subwindows in an image. The capability of WaldBoost is summarized in Table 7.3 reporting the difference in number of weak classifier evaluations between WaldBoost and AdaBoost.

The numbers reported in Table 7.3 are averaged over 2000 images of 1024×768 from the TME dataset. These images contain significant amount of sky, clouds and road which are not of interest to the WaldBoost detector since they are immediately rejected. On average WaldBoost only requires 2.1 weak classifier evaluations per subwindow. In other words, a large amount of unpromising subwindows is already rejected after the first feature evaluation. This is the result that we are hoping for. WaldBoost is very effective in early rejection of regions encountered by the sliding-window. The table also mentions the importance of speeding up the classification of subwindows in an image: we report the number of windows that is evaluated for a single video frame. The speed benchmarks that will be discussed in Section 7.8 show that feature extraction and evaluation of the weak classifiers is expensive for all subwindows, hence the low average number of weak classifiers evaluated using WaldBoost is essential in reaching high frame rates.

Visual proof of the efficient rejection strategy for non-promising subwindows is given as the WaldBoost *attention heatmap* displayed in Figure 7.7. The heatmap visualizes the number of feature evaluations per pixel, in other words it shows the computational power

Table 7.3: Comparison in the number of weak classifiers evaluated for AdaBoost and WaldBoost per subwindow. For AdaBoost the label of each subwindow is determined after processing all $T = 400$ weak classifiers.

	AdaBoost	**WaldBoost**
Windows per image	$2.3 \cdot 10^5$	$2.3 \cdot 10^5$
Tot. features extracted	$9.4 \cdot 10^7$	$4.8 \cdot 10^5$
Avg. stages per window, $\overline{T}_S$	400	**2.08**



(a) **(b)**

(c) **(d)**

Figure 7.7: ***WaldBoost Attention Heatmap.*** For each pixel in the images we visualize the number of weak classifiers that was evaluated. For blue regions the subwindows were rejected almost directly, while for red regions the entire cascade was evaluated. For a video visualization of the WaldBoost attention heatmap constructed for this article, see: `https://www.youtube.com/watch?v=40lITWa1bTY`.

spent per image region. This figure and the video referred to in the caption, beautifully illustrate the effective pruning strategy offered by WaldBoost. From these visualizations we find an interesting link to our feature representation discussion in Section 7.3. We observe that image regions containing horizontal structures are typically not rejected within the first stages but require more feature evaluations before being rejected. In Figure 7.7c this effect is clearly observable in around the overpass.

Learning stage rejection thresholds for WaldBoost depends on proper density estimation of the class-conditional densities $P(H_t(x) \mid y = \pm 1)$ for evaluation of SPRT. These probability density functions are estimated using an independent validation set in each training iteration. In Figure 7.8 we display a number of estimated probability densities for increasing training iteration. Notice that for training iteration $t = 5$ displayed in the upper left no threshold θ_A could be determined using the SPRT. During our experiments we sometimes observed this problematic behavior. In particular for early training stages tight rejection thresholds are important for pruning a large amount of subwindows in the detection phase.

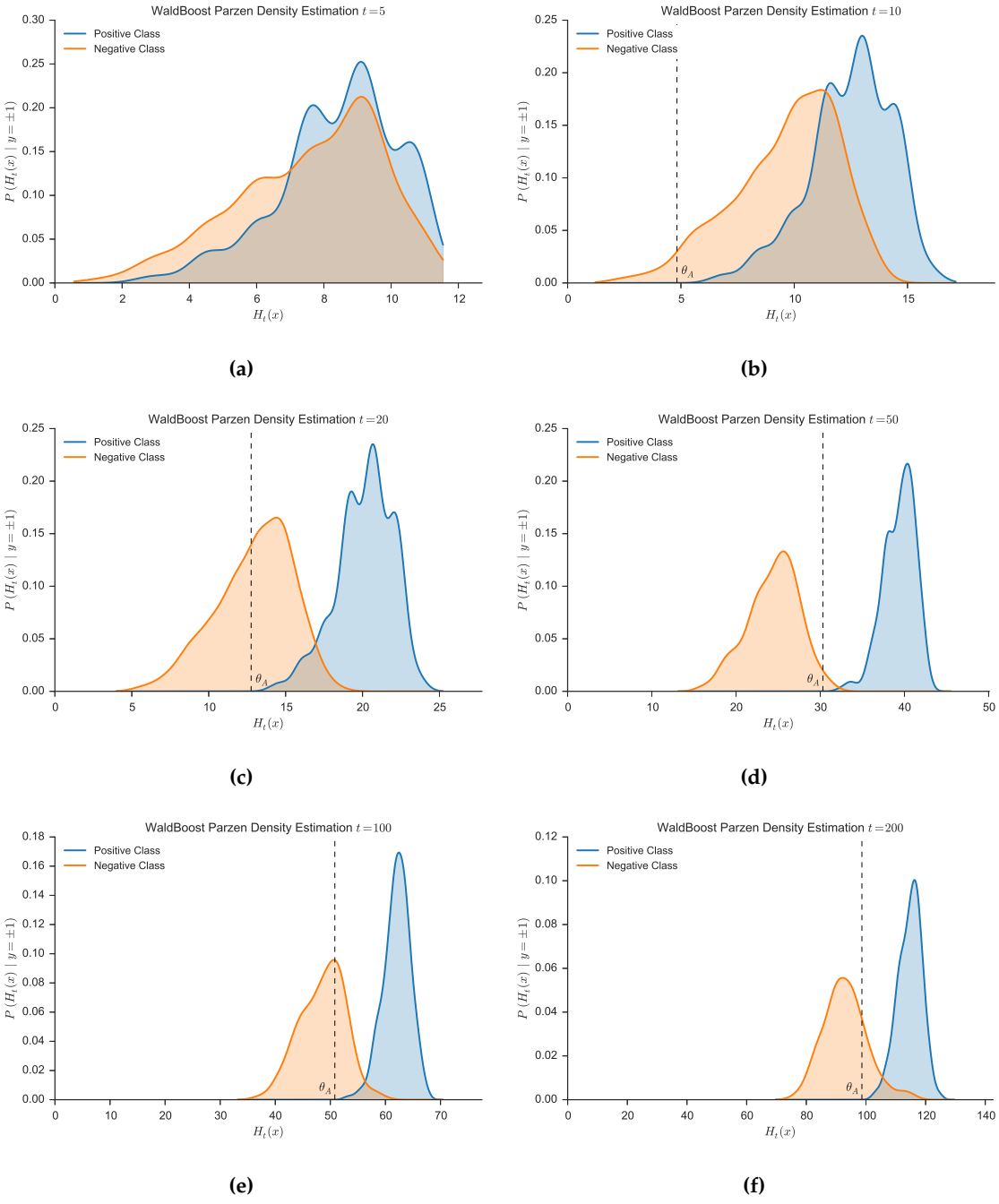


Figure 7.8: ***WaldBoost Non-parametric Density Estimates.*** Class-conditional density estimates for increasing training iteration using Parzen windows on the independent validation set. Dashed lines indicate the position of threshold θ_A determined using the SPRT.

7.5 Experiments on Multiscale Feature Approximations

The experiments in this section focus on the approximation accuracy and application of the power law for feature scaling. Recall from Chapter 6 that the power law for feature scaling, describing the relationship between channel features f_Ω extracted for two different scales s_1 and s_2 is given by:

$$\frac{f_\Omega(I_{s_1})}{f_\Omega(I_{s_2})} = \left(\frac{s_1}{s_2}\right)^{-\lambda_\Omega} + \mathcal{E}. \tag{7.1}$$

This relationship was originally found for image ensembles, although it was later derived for individual images by decomposition in a set of patches [13]. The relationship is an approximation that is expected to hold *on average*. There are several factors that determine the approximation accuracy. For example, we can expect that the approximation breaks down for larger scale differences between s_1 and s_2. Also the relationship will not hold on a per-pixel basis or for small channel features since the idea originates from the scale invariance of natural images which holds for image ensembles. In this section we perform a series of experiments with the goal of obtaining better insight in what features are "stable" in the sense that they can be approximated accurately over a scale range. We feel that the original works on feature scaling [15, 13] fall short in addressing these questions; for example these publications only report the standard deviation of the ratio (6.19) for full 128×64 images. These results are very optimistic since in real world applications smaller feature regions are approximated using the power law, hence resulting in less accurate approximations. With the experiments in this section we attempt to give better insight in the approximation accuracy of channel features.

For the experiments in this section discussing the *error* in the approximation accuracy we use the following methodology. To determine the fidelity of the power law approximations we compare *approximated* channel feature responses to the *exact* responses in our experiments. Given an input image I and some window w indicating the region of a channel feature, the feature response for image I_s at scale s relative to I can be estimated using the power law. The exact feature response for window w_s can be computed by explicitly rescaling image I to obtain I_s, recalculating the channels and evaluating the feature by summing the pixels in the region. For estimating the channel feature as summation of pixels inside a region, we need a slight modification to the expressions introduced in Chapter 6. Due to our redefinition of channel features as global mean in (6.15) we need to include the number of pixels $|w|$ and $|w_s|$ in our calculations. Therefore the approximation relationship for channel features defined as the sum of pixels inside a window $i_s, j_s \in w$ becomes:

$$\begin{aligned} f_\Omega(I_s^{w_s}) &\approx f_\Omega(I^w) \cdot s^{-\lambda_\Omega} \\ \frac{1}{|w_s|}\sum_{i_s j_s} C_s(i_s, j_s) &\approx \frac{1}{|w|}\sum_{ij} C(i,j) \cdot s^{-\lambda_\Omega} \\ \sum_{i_s j_s} C_s(i_s, j_s) &\approx \frac{|w_s|}{|w|}\sum_{ij} C(i,j) \cdot s^{-\lambda_\Omega}. \end{aligned} \tag{7.2}$$

Where the summations are over the rectangular windows at the given scales. This equation is applied to approximate the *sum* of pixels in a rectangular window at scale s without recomputing the channels at this scale. In the following subsections we perform experiments to determine how the approximation accuracy changes for decreasing window size $|w|$ and increasing scale difference s in both upscaling and downscaling directions. To measure the accuracy of the approximations, we define the *relative error* between the exact feature response obtained by scaling the window and the approximation directly from the original scale.

$$\text{relative error } \epsilon = \frac{\left| f_\Omega(I_s)^{exact} - f_\Omega(I_s)^{approximation} \right|}{f_\Omega(I)^{exact}} \tag{7.3}$$

The hypothesis is that relative error increases with increasing scale difference s. Also we expect the error to increase for smaller images or feature windows. In the subsections to follow, we perform experiments to test these hypotheses. The experiments are conducted in the context of object detection using integral channel features, therefore we show the approximation results over all ten image channels (Section 4.1). Performing approximations requires the channel specific scaling parameters λ_Ω and a_Ω. These values are are empirically determined using the approach discussed in Section 6.4.1. Our implementation uses the values summarized in Table 7.4. Since the coefficients are almost identical for all six different gradient orientations we average them. Notice how the power law is linear for color channels given that $\lambda_\Omega = 0$.

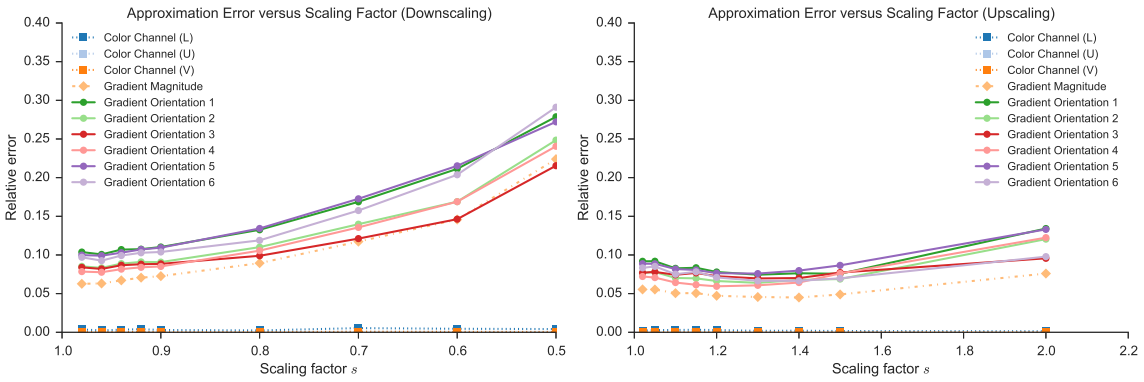
Table 7.4: Channel characteristic scaling parameters (Section 6.4.1).

	Downscaling		**Upscaling**	
	a_Ω	λ_Ω	a_Ω	λ_Ω
Color Channels	1.000	0.000	1.000	0.000
Gradient Magnitude	0.917	0.656	0.920	0.967
Gradient Orientations	0.900	0.591	0.900	0.901

7.5.1 Approximation Accuracy versus Scaling Factor

For this first experiment we are purely interested in the effect of increasing the relative scale difference s on the approximation accuracy. Therefore keep the feature window size fixed to 50×50 pixels and we randomly sample $N = 10\,000$ windows from images contained in the TME dataset. The windows are evaluated for all ten channels, *i.e.* for all windows we compute the corresponding channel feature as summation of pixels inside the window over each channel.

In Figure 7.9 the relative errors over one image octave in upscaling and downscaling direction are displayed. For the LUV color channels we observe an relative error $\epsilon = 0$, which is in agreement with our expectation since the approximation for color channels is exact. This explicitly shows the effectiveness of scaling Haar features in a object detection



(a) Downscaling, $s < 1$

(b) Upscaling, $s > 1$

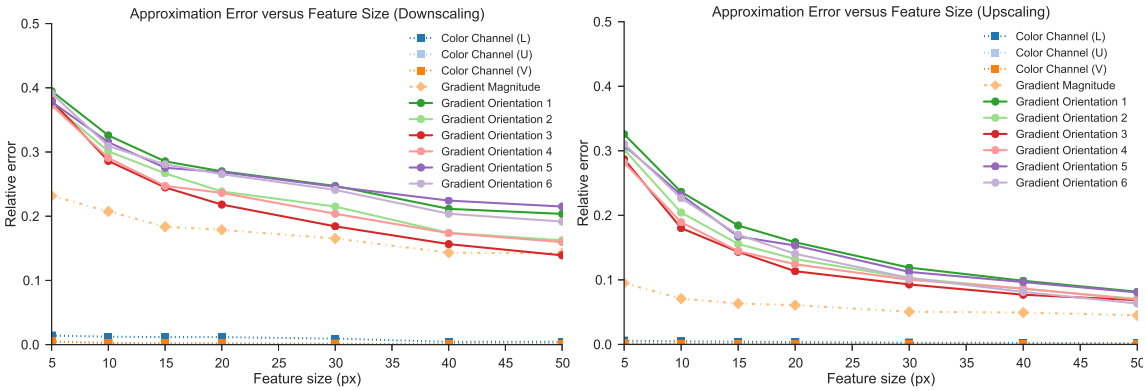
Figure 7.9: ***Approximation Error versus Relative Scale Difference.*** Relative errors between approximations and exact feature values computed for $N = 10000$ windows of size 50×50 sampled randomly from images of the TME dataset.

framework. Only the feature window location, size of the window and classification threshold have to be modified for performing multiscale object detection without rescaling the input image. The difference between the plots for upscaling and downscaling is evident. In agreement with the differences between upscaling and downscaling for rescaling *histograms of oriented gradients* in Figure 6.2b we observe a significantly higher standard deviation for downsampling due to the loss of image content. In the case of upsampling no image content is lost. An important conclusion from this experiment is that the power law breaks down for larger relative scale differences. We cannot expect accurate approximations for scale differences large than one image octave.

7.5.2 Approximation Accuracy versus Feature Size

For the experiment in the last section the feature window size was kept constant at 50×50 pixels. In the current section we study the influence of variation in the window size on the approximation accuracy. The derivation of the power law (6.18) relies on the decomposition of an image into a set of image patches which can be regarded as a small image ensemble. We cannot expect this decomposition to hold for small images, *i.e.* on a per-pixel basis the approximation (7.2) will result in noisy approximations. In Section 4.2.1 the generation of the random feature pool was discussed, also mentioning the requirement that features windows should at least contain 25 pixels. Thus, the feature windows can be significantly smaller than the 50×50 pixels studied in the previous experiment. Given this fact, for the current experiment, we sample uniformly at random square feature windows of a varying size which is the parameter that in this experiment. This time we keep the scales constant, *i.e.* for downscaling we set the relative scale to between the images to $s = 0.5$ while for upscaling we use $s = 2$. The plots showing the relative errors for increasing window size are displayed in Figure 7.10.

We observe higher relative errors in the case of downscaling. This can be explained by the fact that the features are already small at the original scale. When the image is



(a) Downscaling, $s = 0.5$

(b) Upscaling, $s = 2$

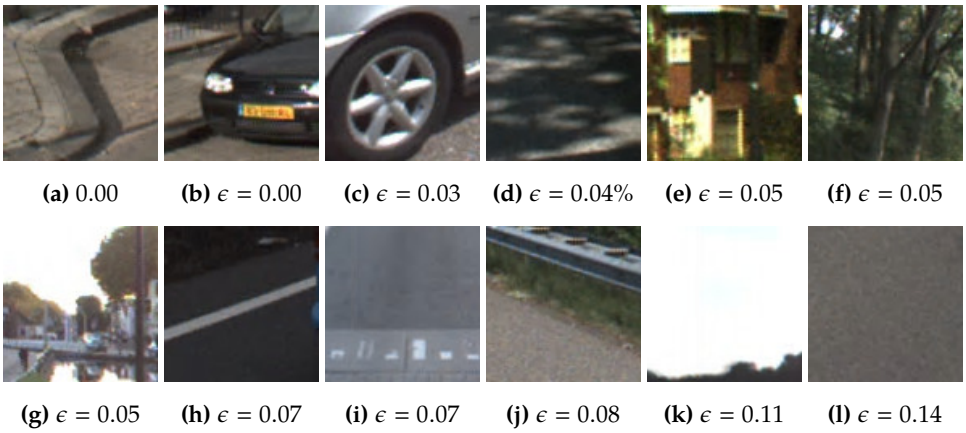
Figure 7.10: ***Approximation Error versus Feature Window Size.*** Relative errors of approximations with increasing feature window size. Feature windows are of square size $w \times w$ where the w is denoted on the x-axis.

downscaled with a factor 0.5 we power law is used for approximating the response over a very small number of pixels, hence the approximation is noisy resulting in a high relative error. Not surprisingly, the results indeed show that approximation error decreases when aggregating statistics over larger image regions. Again the approximation error for color channels is zero. We also observe that curves for upscaling are closer together compared to the downscaling variant. Possible cause for this difference is the removal of pixels in downscaling thus also removing some gradient orientation information which affects only a single gradient orientation channel.

Also the results clearly indicate the difference between gradient magnitude and gradient orientation channels. Gradient orientation channels are more "sparse" than the gradient magnitude channel. Due to the orientation binning, the gradient for a pixel in the original image is assigned to one of the orientation channels. In other words, the gradient orientation channels contain on average 6× as much "empty" pixels. For sparse channels where frequently $f_\Omega(I) \approx 0$ the standard deviation in the ratio $f_\Omega(I_s^i)/f_\Omega(I_1^i)$ will be large. This effect is illustrated in Figure 7.11 which displays the relative approximation error for some randomly selected example windows. Examples with limited meaningful structure show higher approximation errors. We could perform an experiment to study relationship between the average gradient magnitude and approximation error. However in the context of feature selection by Discrete AdaBoost, this problem of sparse channels is not relevant since the feature selection process will choose the most discriminant channel features at each stage and it is unlikely that these features are sparse.

7.5.3 "Approximation Direction" in Object Detector

The two preceding experiments have given us insight in the effect of scaling distance and window size on the approximation accuracy. More important however, is the actual effect of feature approximations on the detection rate of our object detector. Therefore in this section we perform experiments directly related to our object detection framework. As



(a) 0.00 **(b)** $\epsilon = 0.00$ **(c)** $\epsilon = 0.03$ **(d)** $\epsilon = 0.04\%$ **(e)** $\epsilon = 0.05$ **(f)** $\epsilon = 0.05$

(g) $\epsilon = 0.05$ **(h)** $\epsilon = 0.07$ **(i)** $\epsilon = 0.07$ **(j)** $\epsilon = 0.08$ **(k)** $\epsilon = 0.11$ **(l)** $\epsilon = 0.14$

Figure 7.11: ***Relative Approximation Error Examples.*** Relative error in approximation for example windows. Approximations are made for the upscaled image with a scaling factor of $s = 1.25$. The relative errors are averaged over all ten channels. Note that patches with limited structure the approximation error is noticeably higher.

discussed in Section 6.5 our detector performs object detection without image rescaling. This is achieved by training detectors at four different base scales: $s' = 0.5, 1, 2, 4$. During detection phase, the base scales are used as *starting point* for the approximations to intermediate scales. An important aspect of the detector is what base scale to select as starting point for the approximation. As example, consider the scale $s = 1.5$ as current scale of interest in the detector. The approximation can be computed from base scale $s' = 1$ but also $s' = 2$ as both scales s' are at the same distance from intermediate scale s. In the framework proposed by Dollár *et al.* [13], always the nearest base scale is chosen. However experiments in the previous sections have demonstrated differences in relative error for upscaling and downscaling directions. Therefore we are motivated to study two different strategies for choosing the base scale used as starting point for the approximations.

Prior to this experiment our detector always selected the nearest detection scale as proposed in the literature. For a given scale s we could also select the nearest detector smaller than the current scale, meaning that the feature approximations are performed in *upscaling* direction. Likewise we could also choose the detector larger than the current scale corresponding to approximations in the *downscaling* direction. These possibilities are illustrated in Figure 7.12. Orange markers indicate three example scales to consider.

To determine the effect of different approximation strategies on the overall detection rate of our multiscale classifier, we modify the implementation to allow the option of performing feature approximations *always* in the upscaling or downscaling direction. Given this modification we evaluate the detector performance for these settings on the images in the TME dataset. The surprising results are displayed in Figure 7.13. We observe that performing approximations always in the **downscaling direction** has a significant advantage over the other strategies. This is surprising, even more so because the experiments in the previous subsections have shown that performing the approximations in upscaling direction corresponds to lower relative error. Possibly the result is explained

by the fact that performing approximations in downscaling direction means that the approximation is always initiated from a larger trained model. Training models at a larger size uses training examples at higher resolution. Presumably this causes the detectors at larger scale to be better for the classification task, hence resulting in the behavior depicted in Figure 7.13. For the remaining experiments and benchmarks we include the findings of this experiment, *i.e.* we set our detector to always perform approximations in downscaling direction.

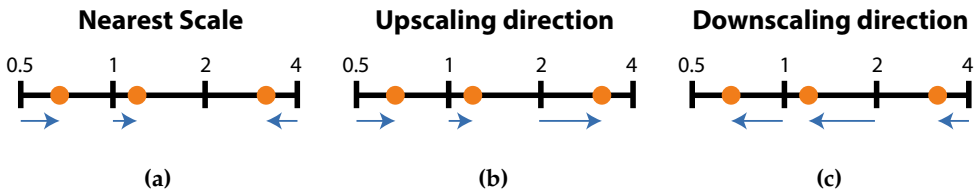


Figure 7.12: ***Feature Approximation Directions.*** Different strategies for choosing the base scale as starting point for the approximations. The base scales are $s' = 0.5, 1, 2, 4$. The feature values for intermediate scales s are computed by approximation are indicated in orange. Blue arrows indicate the direction of the approximation using the power law.

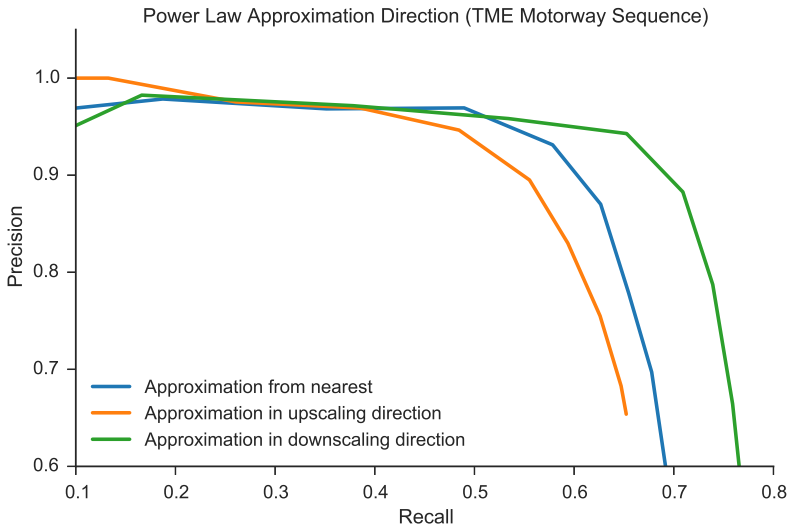


Figure 7.13: ***Detection Quality versus Feature Approximation Directions.*** Comparison of different feature approximation strategies for multiscale approximations using the power law. Performing the approximation always in downscaling direction has a significant benefit over the other two approximation strategies.

7.6 Detector Robustness: *Translation* and *Scaling*

In this section we perform a number of experiments on the "robustness" of our baseline detector. We train a rear-view car detector and study the robustness of our detector with respect to *translation* and *scaling*. To some extend these results motivate the parameters in our final classifier: the sliding-window step size in both directions and the number of scales per octave. For both parameters there is a trade-off between the detector speed and detection quality. Notice that instead of performing the experiment in this section, alternatively we could have changed these parameters directly in our detector and study the influence on the detection rate over our test set. These results are given in the next section. However we believe that the results presented in this section give a more intuitive understanding of the translation and scaling robustness hence we decided to include them.

7.6.1 Detector Robustness to Translation

The object detector is implemented as binary classifier using the sliding-window technique. Moving across the image the sliding-window scans the image by feeding the descriptors of each subwindow to the classifier. The specified sliding-window step size at which the detector scans across the image is an important parameter influencing both speed and detection rate. Increasing the step size results in higher risk of missing target objects because they are not aligned inside the sliding-window for any position. However a step size of single pixel corresponding to evaluation of *all* subwindows is not feasible for detectors aiming to operate in real time. There is thus an important trade-off between speed and detection rate for sliding-windows step size.

In this experiment we study the effect of misaligned objects inside the sliding-window. For this and the next experiment we extract ground truth bounding boxes from the TME dataset. These are annotations from full images which allow us to extract the bounding boxes with padding. First we extract 500 ground truth bounding boxes and *translate* them with a varying amount of distance in any of the four directions. The number of pixels and direction of translation are the parameters in this experiments. As result of the translation the objects are misaligned within the bounding box. To study the effect on the detection rate, we apply our baseline detector of 400 weak classifiers to the shifted bounding boxes and determine the detection rate over all 500 samples. For example we translate all bounding boxes 4 pixels to the left and study the change in detection rate in comparison to the bounding boxes with "perfect" alignment corresponding to a translation of zero pixels. An example image indicating the translation in all directions is displayed in Figure 7.14.

Figure 7.15 exhibits the detection rate for increasing translation in all directions. We compare the detection results of shifted bounding boxes to the original detection rate of 97.9% over these 500 images. We observe a distinct difference between translation in horizontal and vertical direction. The detection rate suffers more from horizontal

(a) Original **(b)** Left **(c)** Right **(d)** Up **(e)** Down

Figure 7.14: ***Translation of Bounding Boxes.*** Relative bounding box translation of 8 pixels with respect to the target object (ground truth). Note that directions indicate shift direction of the bounding box, not the direction of translation for the object.

translations. This effect might be explained by the fact that the aspect ratio for cars typically is $w/h > 1$. Since the bounding boxes are all of square size, there typically is more padding on top/bottom of the car compared to padding on left/right. Thus a small translation in vertical direction is not likely to remove part of the car, which is not true for translation in horizontal direction due to the narrow padding. This being said, the experimental results motivate us to set the sliding-window step size larger in vertical direction. For the speed benchmarks later in this chapter, the step sizes are set to 8 and 10 pixels in horizontal and vertical direction respectively.

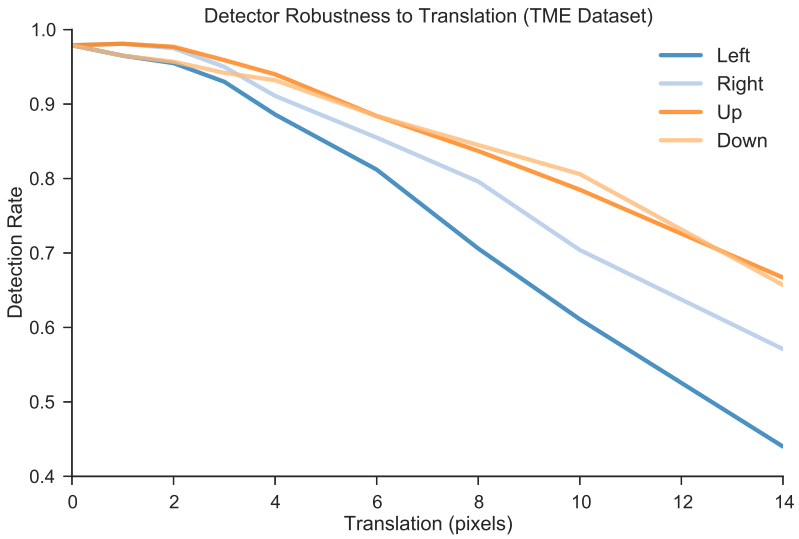


Figure 7.15: ***Robustness to Translation.*** Effect of translating ground truth bounding boxes in all directions, reflecting the effect of varying sliding-window step sizes. Detection rates are computed over 500 positive test examples.

7.6.2 Detector Robustness to Scaling

In similar fashion to the experiment of the previous section, we are also interested in scale robustness of our detector. Scaling of the bounding boxes relates directly to the number of scales per octave that we should analyze for full images. Again we use the bounding boxes extracted from the TME dataset, however this time we scale the bounding boxes with a factor s. Scaling factors $s > 1$ correspond to increasing the bounding box, thus

introducing more padding around the object. Likewise $s < 1$ indicates shrinking the bounding box, removing content of the bounding box, *i.e.* "zooming in". This process is illustrated in Figure 7.16. We scale the bounding boxes 20× in the range $s = [0.5, 2.0]$ and study the decrease in detection rate over all 500 test examples. The effect of rescaling on the detection rate is established in Figure 7.17. Not surprisingly the detection rate drops significantly faster for downscaling since the bounding box size is decreased and part of the object is cropped. The detector is remarkably robust to upscaling bounding boxes for small scaling factors. Up to a scaling factor of $s = 1.3$ the classifier still predicts correctly 95% of all bounding boxes. For scaling factors $s > 1.3$ we observe a steep decline in the detection rate.

(a) $s = 2.0$ **(b)** $s = 1.6$ **(c)** $s = 1.0$ **(d)** $s = 0.8$ **(e)** $s = 0.6$

Figure 7.16: ***Scaling Bounding Boxes.*** Ground truth bounding boxes of TME Motorway Sequence rescaled with factor s. Center indicates the original size.

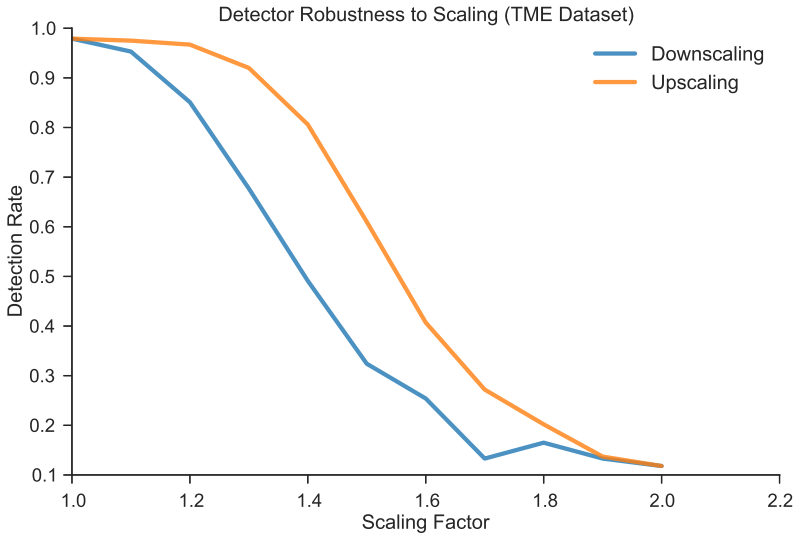


Figure 7.17: ***Robustness to Scaling.*** Effect of scaling ground truth bounding boxes on detection rate of baseline classifier. For downscaling the x-axis plots $s' = 1/s$. Detection rates are computed over 500 positive test examples.

7.7 Overall Detection Quality

We now arrive at the conclusive section on the detection quality of our rear-view car detector. In this section we report final detection results of our overall detection framework on the TME dataset. Also we report the effect of WaldBoost and feature approximations on

the detection rates in comparison to our baseline detector. The evaluation methodology for this section is as follows. The multiscale detector is tested on the TME sequence with manually corrected bounding boxes. This dataset includes 465 images of size 1024×768 with a total of 1298 annotated cars. Detections are compared to the ground truth boxes and regarded as true positive if the PASCAL VOC overlap criterion exceeds 0.5 (details are given in Section 3.2.1). Before reporting final detection results, we determine the optimal parameters in three experiments.

Number of Weak Classifiers. In Figure 7.18 we plot precision-recall curves for variation in the number of weak classifiers. All curves are obtained using the same multiscale detector. Only the first weak classifiers are enabled, the rest is disabled for the experiment. Surprisingly adding more than 300 weak classifiers does not seem to increase detection rates. Instead we observe *overfitting* when using more that 400 weak classifiers.

Sliding-Window Step Size. The next experiment demonstrates the relationship between the sliding-window step size and the detection rate. The results displayed in Figure 7.19 show that choosing step sizes larger than 8 pixels in both direction has a negative effect on the detection rate. Notice that the performance for a step size of 6 pixels is worse that for a step size of 8. We are unable to explain this strange observation, it might be that by coincidence the detector misses a number of observations. For the remaining experiments we set our sliding-window step size to 8 pixels in both directions.

Number of Scales per Octave. The detection quality decreases dramatically when analyzing less than 4 image scales per octave as is revealed in Figure 7.20. For the remaining experiments we set the total number of scales to 25.

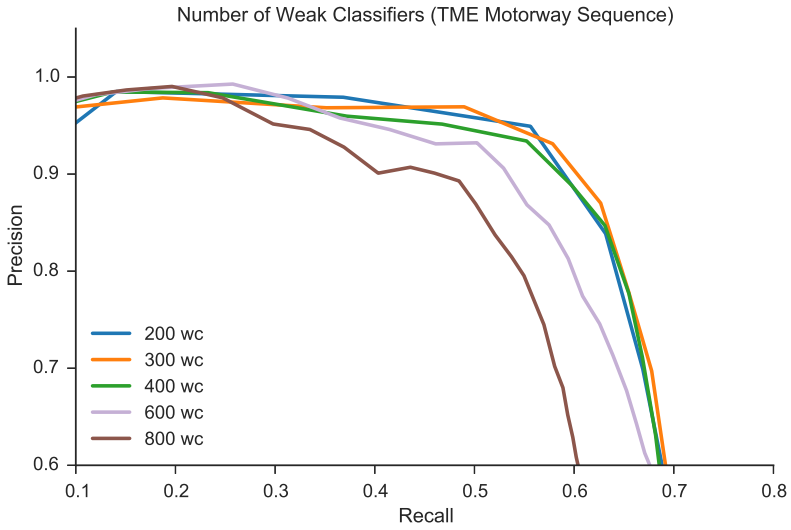


Figure 7.18: ***Number of Weak Classifiers.*** Adding more weak classifiers to the object detector beyond 400 has does not increase detection performance. Statistics are computed over the TME dataset with a sliding-window step size of 8 pixels.

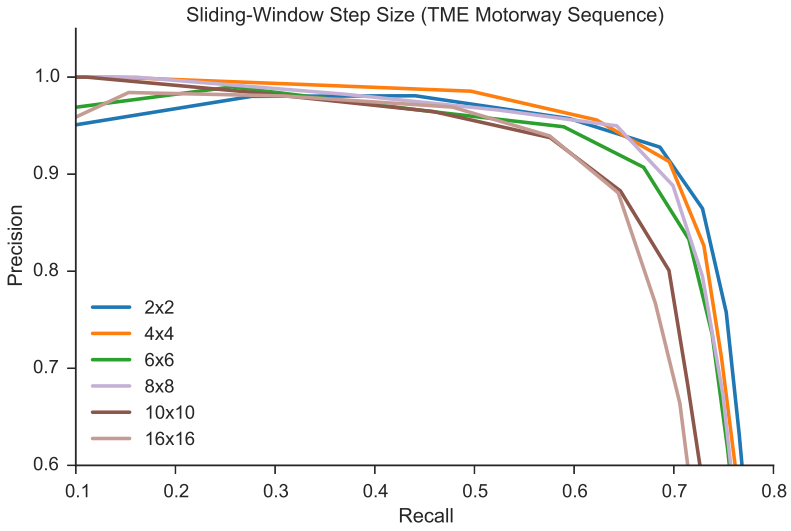


Figure 7.19: ***Sliding-Window Step Size.*** Detection rate versus sliding-window step size. Results are determined over the 465 test frames. The strong classifier contains 400 weak classifiers.

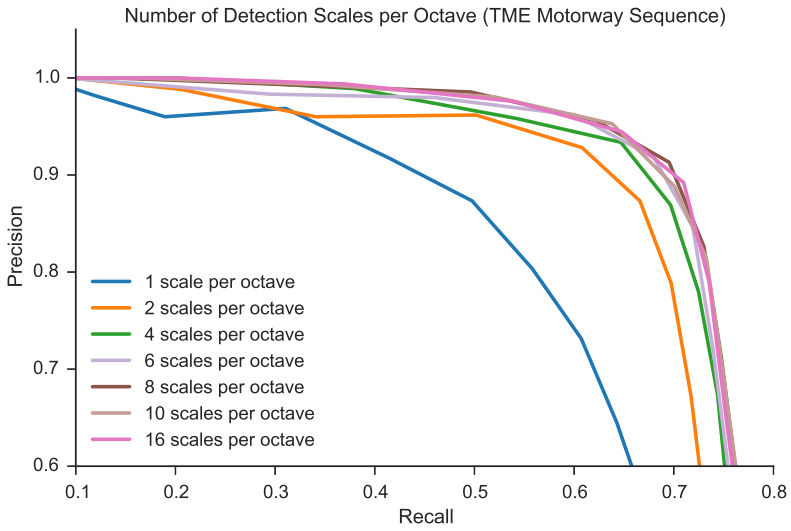


Figure 7.20: ***Detection Scales per Octave.*** Detection rate versus number of scales per octave. Results are determined over the 465 test frames. The strong classifier contains 400 weak classifiers.

7.7.1 Comparison in Detection Quality

Given the set of optimal parameters determined in the preceding section, we perform the final experiment on the detection rate demonstrating the performance of our overall detection framework. Additionally we show the performance for each individual component of our detector. First we report detection quality of our baseline detector and then include WaldBoost and feature approximations to show their effect on the detection rate. As comparison we also benchmark our detector against the deformable part-model detector of Felzenswalb *et al.* [24]. For this comparison we use the OpenCV implementation of the DPM detector. The comparison is somewhat unfair to the DPM detector since this detector is trained on the PASCAL VOC dataset whereas our detector is trained on the TomTom dataset. Evaluating the DPM detector over all 465 images of size 1024×768 took a significant amount compared to our own detector. Processing time are about 2 seconds per image for the DPM detector. The detection quality of the DPM detector should be seen only as rough estimate. We are unable to make a prediction on the detection rate of the DPM detector with proper settings and training procedure.

Detection Rate Comparison. The final set of precision-recall curves illustrating the overall detection performance and the per-component comparison is given in Figure 7.21. The first observation we make is that WaldBoost only has a minor influence on the detection rate compared to the baseline detector using AdaBoost. Apparently a small number of rejection thresholds are too tight, resulting in the loss of true positive detections. More interesting however, is the increase in detection rate when enabling feature approximations for multiscale detections without image rescaling. The multiscale feature approximations are enabled on top of WaldBoost, hence we would also expect a small decrease in detection rate caused by the feature approximations. However this is not the case, as we observe a small increase in detection rate due to the use of multiscale approximations. We explain this behavior by the fact the multiscale detector is trained at different scales, hence resulting in a more accurate detector for each scale. In other words, since a detector for all four scales is trained, each classifier is better suited for the classification task at that particular scale. A detector trained at larger can use fine-grained details as feature in comparison to detectors trained at smaller training examples. Therefore the features are better suitable for the detection task over a wide range of image scales. Training the model at a number of scales thus has an beneficial effect on the overall detection rate.

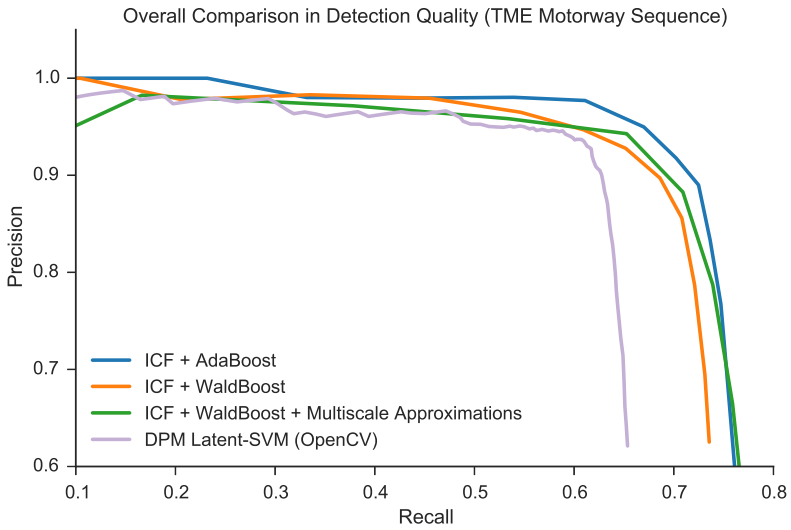


Figure 7.21: ***Overall Comparison in Detection Rate.*** Starting from the baseline detector, we show the effect of adding WaldBoost and feature approximations to the detector. All detectors contain 400 decision stumps and the detector is applied with a sliding-window step size of 8×8 over 25 scales.

7.8 Speed Benchmarks

In this final section of the experiments chapter we report overall detection times and analyze the per-component contribution to the processing time per image. Due to the parallel implementation on both CPU and GPU, performing time measurements is not straightforward. Time measurements themselves also influence the reported times, hence the absolute processing times are inaccurate. In particular for time measurements on per-component basis the absolute processing times are not accurate, however the relative time measurements give a good impression of what components are the bottleneck. First we report the processing times of extracting integral channel images, working gradually towards our final detector with adding components on the way. All time benchmarks reported in this section are performed on `Platform I`.

Channel Extraction Runtime. The first analysis shows the required time for extracting integral channels given our GPU implementation. Table 7.5 summarizes the time required for extracting image channels with increasing number of levels in the image pyramid. As mentioned in Section 4.4.1 our implementation uses unsigned gradient orientations displayed in Figure 4.2. These are extracted using simple one-dimensional filter kernels without prior smoothing. Extracting all 10 channels using our GPU implementation runs at 80.6 fps for 640×480 images. For 1024×768 the speed decreases to 40.0 fps. This is almost five times as fast as the original implementation [16] reporting channel extraction running at 40 fps on 320×240 images, however the authors do not mention hardware specifications.

Table 7.5: ***Channel Extraction Time.*** Total time required for computing integral channels for 640×480 images with increasing number of scales. Image octaves are $\{0.5, 1, 2\}$, remaining scales are intermediate. Results are averaged over 200 frames.

Scales per Octave	Total # Scales	Time (ms)	FPS
-	1	12.4	80.6
1	3	80.7	12.4
2	5	124.7	8.02
4	9	211.1	4.74
6	15	389.8	2.56
8	17	402.8	2.48
10	21	492.3	2.03

Comparison Runtime AdaBoost versus WaldBoost. To summarize the differences in feature extraction and classification time between AdaBoost and WaldBoost we outline benchmark results in Table 7.6 and Table 7.7. We emphasize that performing time measurements is not straightforward since processing the individual subwindows is executed in a parallel loop. The processing times indicate the total time spent over all threads. In other words, the actual computation time is about 4× lower since `Platform I` runs four parallel CPU threads. The difference in both feature extraction time and classification time is enormous. At four scales per octave the speed-up is 200× for feature extraction and 240× for classification. This speed-up is completely explained by the early rejection of subwindows for classification. AdaBoost performs feature extraction for all 400 weak classifiers and evaluates the decision stumps, while for WaldBoost on average only 2.1 features are extracted and classified.

Per-Component Runtime Analysis. The most informative insight in the per-component contribution to the overall runtime is given in the pie charts displayed in Figure 7.22. This illustration summarizes our motivation for extending our baseline detector with WaldBoost and multiscale approximations. For our baseline detector we observe that per frame approximately 62% of the time is spent on classification of subwindows. Together with feature extraction, both components account for 93% of processing time. This analysis is our foremost argument to implement WaldBoost for speeding up classifications of subwindows. As we can observe from the pie chart corresponding to WaldBoost, the time contributions change completely. The time spent on feature extraction and classification drops dramatically, with as result that channel computation dominates the classification time per image. Consequently, this demonstrates the importance of implementing multiscale approximations without image rescaling. The pie chart for our overall detection framework is given in the bottom. Notice how the per-component contributions are perfectly in balance for our overall detection framework. Extracting integral channels consumes about 28.6% of the processing time per image, while feature extraction requires 29.2% and classification of subwindows 42.2%.

Overall Contributions to Speed-Up. The final contributions to speeding up our final classifier are given in Table 7.8. We also report runtimes with a reduction of the search space to show impressive speeds and for comparison with the work of Beneson *et al.* [3]. Results are given in Table 7.9, which also includes four videos showing our overall detection framework in action. We acknowledge that not all runtime reports are in agreement with each other, which is mainly due to time measurements. The overall time measurements without intermediate measurements reported in Table 7.8 and Table 7.9 are the most accurate.

Table 7.6: ***AdaBoost Classification.*** Total time required for feature extraction and AdaBoost classification for 640 × 480 images with increasing number of scales. AdaBoost classifier contains 400 weak classifiers. Averages over 200 video frames.

Scales per Octave	Total # Scales	Feat. Extraction (ms)	Classification (ms)
1	3	1463.6	2863.8
2	5	2091.0	4281.1
4	9	3545.1	7210.9
6	15	6363.6	12908.3
8	17	6482.3	13293.8
10	21	7887.3	16173.8

Table 7.7: ***WaldBoost Classification.*** Total time required for feature extraction and WaldBoost classification for 640 × 480 images with increasing number of scales. AdaBoost classifier contains 400 weak classifiers. Averages over 200 video frames.

Scales per Octave	Total # Scales	Feat. Extraction (ms)	Classification (ms)
1	3	6.4	10.9
2	5	9.8	16.6
4	9	17.9	30.2
6	15	37.2	61.7
8	17	39.7	65.1
10	21	48.0	78.9

Table 7.8: ***Contributions to Speed-up.*** Speed benchmark over 640 × 480 images analyzed at 30 scales. Runtimes are averaged over 200 video frames. For all settings the strong classifier consists of 400 weak learners.

	Relative Speed	Absolute Speed
Baseline (ICF + AdaBoost)	1.0×	1.3 fps
+ WaldBoost	2.6×	3.2 fps
+ Multiscale Approximations	43.1×	56.0 fps

Table 7.9: ***Search Space Reduction and Videos.*** Examples of our object detector running on video sequences. The runtimes indicate the speed of our detector running on `Platform I` over the given search space with a sliding-window step size of 8 pixels in both directions. The videos are slowed down to 50 fps for displaying.

Sequence	Search Space	Speed	Video
Eindhoven Airport 1	$640 \times 150 \cdot 25$ scales	148 fps	`https://youtu.be/OG_ZliRPN3c`
Eindhoven Airport 2	$640 \times 150 \cdot 25$ scales	148 fps	`https://youtu.be/MMedzoJvxp4`
TME Sequence 17	$800 \times 300 \cdot 25$ scales	84 fps	`https://youtu.be/lvMJU_rwXOY`
TME Sequence 42	$800 \times 300 \cdot 25$ scales	84 fps	`https://youtu.be/w37jT8AR6Z4`

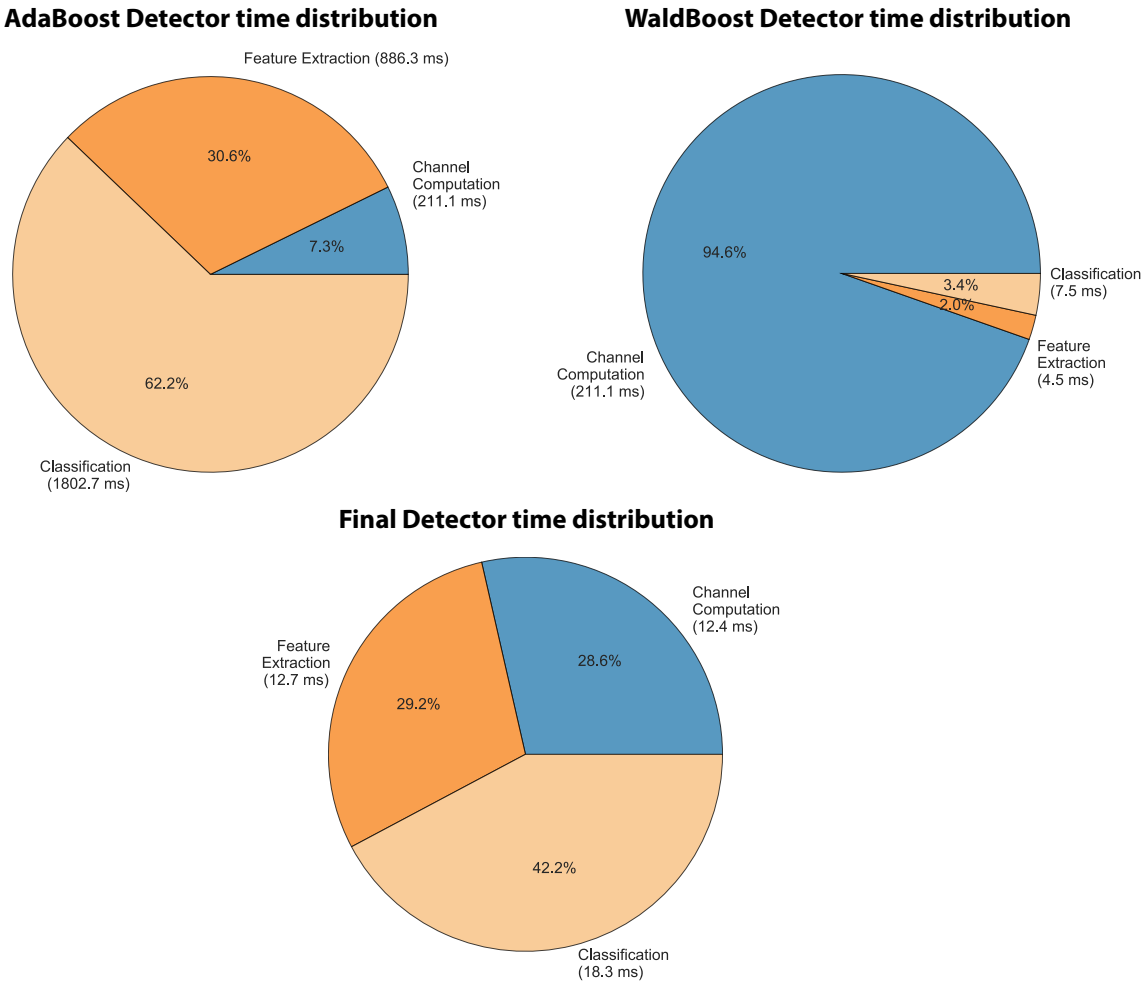


Figure 7.22: ***Time Contribution Per Component.*** Comparison between AdaBoost, WaldBoost and Multiscale Detector. Statistics are computed for classifier consisting of 400 weak classifiers. Processing times are for 640×480 images at 4 scales per octave. Absolute processing times are not accurate due to time measurements.

(a) (b)

(c) (d)

(e) (f)

(g) (h)

Figure 7.23: ***Correct Detections.*** Examples of **good detections** by final object detector. Blue detections are *exact*, while yellow detections indicate detections made by feature *approximation* using the power law.

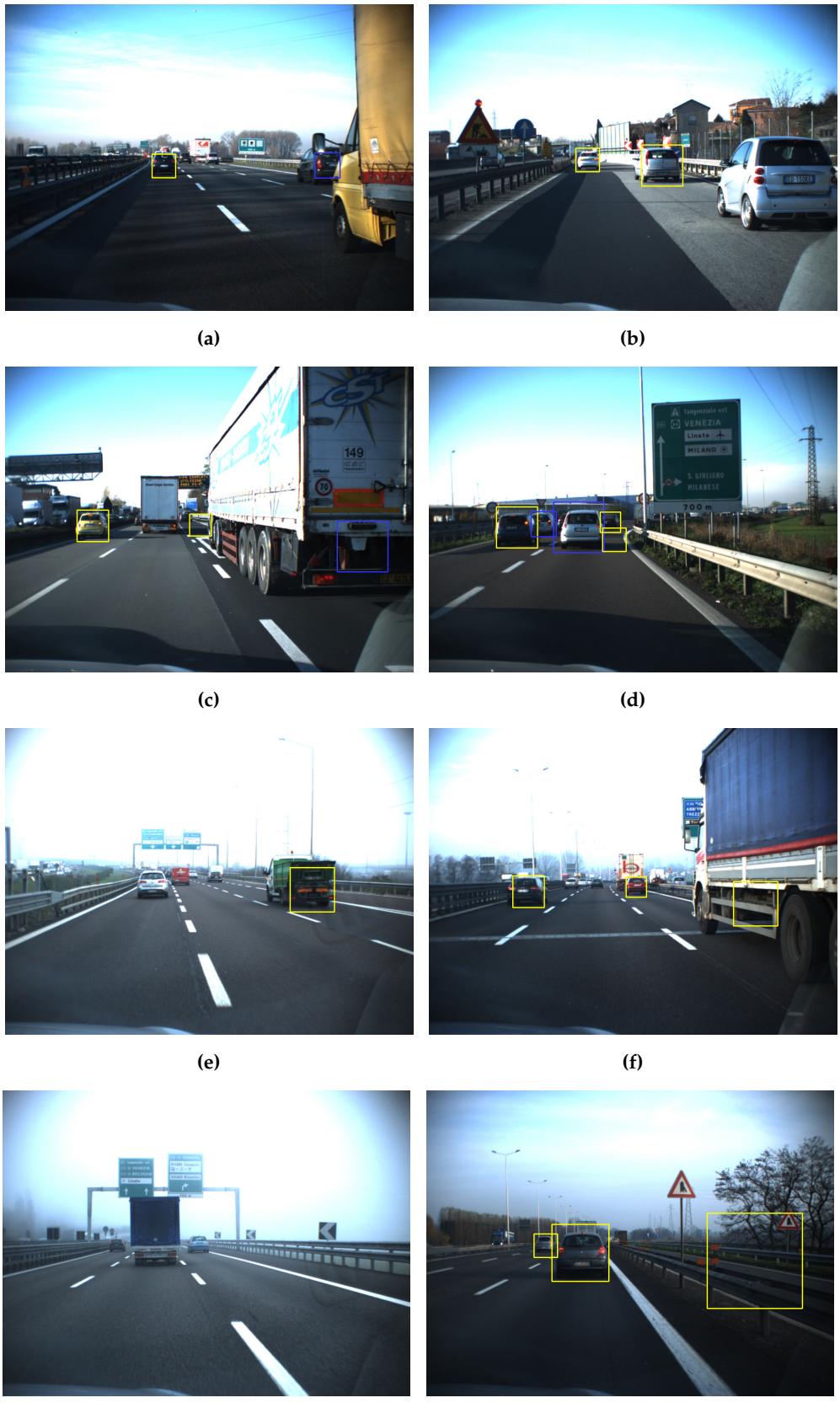


Figure 7.24: *Incorrect Detections.* Examples of bad detections by final object detector. Blue detections are *exact*, while yellow detections indicate detections made by feature *approximation* using the power law.

(a) FN **(b)** FN **(c)** FN **(d)** FN **(e)** FN

(f) FN **(g)** FN **(h)** FN **(i)** FN **(j)** FP

(k) FN **(l)** FN **(m)** FN **(n)** FN **(o)** **FP**

Figure 7.25: ***Incorrect Detections Baseline Detector***. False negative and false positive examples of our final detector. Note that the detector performs poorly for detections in which the image is blurry or distorted. The false positive in the lower right clearly shows that our detector triggers on the shadow and the horizontal structure near the center of the image.

8

Discussion, Future Work and Conclusion

In this research we have introduced a novel object detector with the focus on achieving the highest possible detection speeds without sacrificing on detection quality. We have built upon our baseline detector using integral channel features in a learning framework of Discrete AdaBoost. With our fast GPU implementation of the baseline detector as starting point we have suggested and implemented two approximation techniques for significantly increasing detection speeds, enabling real-time performance. The first improvement focuses on decreasing classification time of individual subwindows of an image by formulating the classification as sequential decision process using WaldBoost. The second approximation technique uses the scale invariance property of image statistics in natural images for better multiscale handling. This allows for fast multiscale object detection without image rescaling. While these ideas are not new itself, we are the first to combine them in a full object detection framework. We show that the combination of these components is very powerful, allowing us to report one of the highest detection speeds available in computer vision literature while maintaining excellent detection quality.

Our detector runs at approximately 56 fps over $640 \times 480 \cdot 30$ scales on a modern laptop. Speed benchmark results and per-component speedup are given in Table 7.8. Not surprisingly, reduction of the search space increases detection speed significantly. Excluding the top and bottom part of the video frames, effectively reducing the search space to $640 \times 150 \cdot 25$ scales, our detector runs at 148 fps. Since objects of interest only appear within this region the detection quality is unchanged. Our object detector is trained on our TomTom car dataset containing a total of 4000 training examples equally divided in positive and negative examples. For evaluating the detection quality of our final detector and the per-component effect on the detection rate we report results on the TME Motorway Sequence containing 465 video frames with a total of 1298 annotated vehicles. For this dataset we report a hit rate (recall) of 0.75 at a positive predictive rate (precision) of 0.9. These results are presented as precision-recall curves in Figure 7.21.

In the final chapter of this thesis we elaborate on the experimental results from the previous chapter in the form of a discussion. Since the broad scope of our project we cannot discuss all experimental results and aspects of our detector but rather we select, what we consider, the more important ones for discussion and future work. Supported by the figures and numbers presented in the previous chapter we motivate what we think are the most promising directions for future research to improve the high-speed object detector we propose. After the discussion and future work we summarize the overall conclusions of our work. In the last two sections, which can be regarded as an epilogue, we briefly address some remarks with the current research and give a short personal reflection on our work.

8.1 Discussion

Detection Speed Comparison. Generally referred to as one of the fastest object detectors available in computer vision literature is the `VeryFast` detector of Beneson *et al.* [3]. We compare our detection speeds to their detector that is designed for pedestrian detection. The authors use stereo-vision to build a "stixel world" for reducing the search space dramatically. Their stixel world reduces the search space to 640×60 pixels $\cdot$ 10 scales. At this particular setting the authors report speeds of 135 fps on a high-end laptop featuring Intel Core i7 and dedicated GPU (faster than our hardware). The ground plane estimation method itself runs at 300 fps on a CPU hence this component is not the bottleneck. As comparison we also reduced the search space by simply excluding top and bottom part of the image to 640×150 pixels $\cdot$ 25 scales. After this search space reduction our detector runs at approximately 150 fps without sacrificing on detection quality. We are comfortable claiming that our object detector is one of the fastest reported in computer vision literature.

Detection Quality Comparison. With respect to the detection quality of our detector we find it difficult to give a comparison to other detectors. Since we are using our own dataset specifically designed for this research, no comparisons to existing detectors are available. One straightforward method for comparing our detector to available literature is by training our detector on pedestrian datasets such as INRIA [10] or Caltech-USA [19] and evaluation on the ETH sequence [21]. These are benchmark datasets for pedestrian detection and are commonly used in computer vision literature. We leave this comparison for future work. In Figure 7.21 we report the detection rate of the pre-trained Felzenswalb DPM detector on the TME dataset. While the training process and settings might not be optimal, it is interesting that the recall rate does not exceed 65% whereas our detector outperforms DPM with about 12% in recall rate. This is some indication that the evaluation sequence is not a simple classification task due to occlusions, distant cars, lighting conditions and the large diversity in vehicle appearance. Given the reported detection rates and the comparison with DPM we are satisfied with the results of our detector on this seemingly difficult evaluation dataset.

Channel Feature Representation. Experiments in this thesis show the importance of vertically oriented gradients for rear-view car detection. The channel importance weight

distribution in Figure 7.3 illustrates that over 25% of the weight in the final classifier is devoted to weak classifiers corresponding to features from that particular channel. Without a doubt this is a surprising result, since it suggests that we can approach the car detection problem using only horizontal edges as descriptors. While visual inspection of the channels confirms that horizontal edges are indeed a strong indication for the presence of a car, the observation that more than one-fourth of the features is extracted from this channel is surprising. We think this observation can be explained partially by misalignment of the examples in our training dataset. The outlines of the cars are not aligned properly, resulting in a limited number of features focusing on the overall shape of the car. However the center region of the training examples tends to contain a large amount of horizontal structure around the license plate, rear window and tail lights. Based on this observation we hypothesize that by improving the alignment of samples in our training data the feature selection process picks more features focusing on the outer contour of the car. This could result in a better balance between the channels selected during the training process, and consequently have a positive influence on the overall detection rate. This is idea is reinforced by the observation that we sometimes observe positive detections for image regions that contain horizontal edges.

Refreshing the Feature Pool. The learning process of our baseline detector includes a factor of randomness with the initialization of the feature pool. For most of our experiments we constructed a candidate feature pool of 40 000 features within a model size of 64×64. This number is by no means exhaustive. Using too many features increases training time significantly and puts too much load on available memory. On the other hand, choosing insufficient size of the feature pool causes detector performance to degrade. Unfortunately, the space of possible feature rectangles within a model of 64×64 is huge. This becomes even more problematic for training detectors at larger scales with model sizes up to 256×256. To overcome this problem, our implementation has the option to (partially) *refresh* the feature pool after a number of predefined iterations in the training process. While we do not report experiments on the effect of refreshing the feature pool, what we observe for training our car detector is that an even larger fraction of features is selected from the gradient orientation channel corresponding to horizontal edges. An overview of feature pool initialization strategies is given in [4], however it seems like we are the first to implement the idea of partially refreshing the feature pool during the training process.

Distant Cars. Our detector achieves approximately 75% detection rate on the TME dataset. We have not performed a quantitative study on the type of detections that are correct hits and what kind of ground truth annotations are missed by our detector. However throughout the research we have made a number of observations by looking at the detection running over video sequences. Visual inspection of our detections on the TME dataset show that the majority of false negative (missed) detections correspond to small annotated objects, *i.e.* cars in the far distance. The size distribution of annotated bounding boxes in Figure 3.6 illustrates that a large fraction of the cars are at distance. This suggests a simple experiment of plotting the precision and recall rates versus the

width of ground truth objects in the dataset. Our hypothesis is that the detector fails to detect a large fraction of cars in the distance. In our overall detection framework the smallest detector is trained at the scale corresponding to a model size of 32×32. However the size distribution of Figure 3.6 reports that more than 20% of the cars in the dataset has a width of less than 30 pixels. We predict that adding a smaller training scale to our learning process increases the detection rate, although we wonder if the resolution of the evaluation images is sufficient for detection of the bounding boxes at such small sizes. Due to pixel discretization the cars not only appear small but also blurry at smaller scales.

WaldBoost. The experiments on WaldBoost clearly show its effectiveness as soft-cascade and we feel like its validity is evident leaving less for discussion. We stress the importance of including a soft-cascade approach for high speed detections, the early rejection of non-promising subwindows is decisive in frame rate. Experiments in Section 7.7 show that WaldBoost rejection thresholds have little negative effect on the overall detection quality of our detector. We report a decrease in recall of about 3% due to rejecting positive subwindows during the detection process. During the course of this research numerous WaldBoost detectors were trained on our TomTom dataset. In some cases we observed that setting the rejection thresholds using the SPRT was impossible for early stages. This is destructive to the speed of the detector, since in particular for early stages it is crucial to reject the majority of subwindows. With no thresholds available, rejection of subwindows is not possible. This problem is caused by the empirical risk introduced with estimating the class-conditional densities. While WaldBoost has statistical validity, the method is by no means simple. The empirical risk introduced by estimating the class-conditional densities displays itself in the difficulty of determining the first few rejection thresholds. Given the difficult strategy of settings the thresholds using the SPRT we wonder if a much simpler strategy could result in similar results. From the density estimates in Figure 7.8 we observe that the threshold θ_A for each stage is located directly at the left tail of the distribution for the positive class. This makes us question why we are using such a difficult strategy for learning the thresholds. A simpler approach could be to keep track of trace scores $H_t(x)$ for an independent validation set and choose the rejection thresholds such that no validation examples are classified incorrectly. However this strategy is more conservative than using the SPRT and will most likely result in a slower detector.

Power Law Approximations. Prior to incorporating multiscale approximations in our detector we were very skeptical of the idea. We were pleasantly surprised by the performance of the power law for feature scaling. Preceding the experiments we had no doubt that the power law relationship (6.14) holds for *ensembles* of natural images. However we were very skeptical with regard to the decomposition of *individual images* to derive the power law for feature scaling by Dollar *et al.* [13]. After performing numerous experiments it turns out that the approximations are indeed a very powerful instrument for performing high-speed object detection without image rescaling, and that our skepticism was incorrect.

Training of Multiple Models. Given our parallel implementation of the training process we train a single scale detector in less that 40 minutes. This opens up some interesting

possibilities in a framework that uses multiple detectors of different scales. At our speeds, running the training process for one night allows us to train more than 15 detection scales. Our experiments show that training the detector at four different scales has a positive influence on the detection rate. We wonder what the optimal trade-off is between adding even more detector scales and the detection quality of the detector. This also raises the question of the general better approach for object detection. We can train a model with "expensive" and complex features that are to some extend scale invariant, or on the other we can train multiple models using simpler features. Multiple simpler models have the advantage that the features are specifically designed for a given scale positively contributing to the overall detection quality. However one of the possible problems with training detectors at multiple scales is the availability of high-resolution training examples. For our TomTom dataset the training examples are available at 80×80. However our largest detector is trained at size 256×256 thus training images must be upscaled first. Therefore the detector at largest scale is trained on blurry images which has a negative effect on the training of larger models. In the ideal case, training examples should be available at the highest possible resolution although we recognize the difficulty of putting such a dataset together. We should be aware of details such as increasing the candidate feature pool and the minimum size of feature rectangles for training models at larger sizes. In our current implementation these parameters cannot be modified.

Feature Approximation Direction. One of the more interesting experiments reveals the optimal direction for performing feature approximations. Somewhat to our surprise, always selecting the closest base scale that is *larger* than the current detection scale results in the superior detection rate as reported in Figure 7.13. Choosing the larger base scale as starting point corresponds to performing the approximation in *downscaling* direction. The detector trained at larger base scale typically learns larger channel features, *i.e.* larger rectangular regions. For feature approximation, these rectangular regions are downscaled and the channel feature is computed as summation of pixels in the corresponding channel. On the contrary, the other approximation strategy uses smaller features that are upscaled. Figure 8.1 shows an illustration of both approximation directions. Larger models are trained at high-resolution images, hence features can focus more on smaller details of the object. In the approximation strategy that performs best, features encoding these details are then downscaled hence losing their information. Therefore we are unable to explain the fact that this approximation strategy performs best. Intuitively we would predict that performing the approximation in upscaling direction yields better results. Since in this case the smaller detector focuses on coarser structure of the objects, upscaling the features maintains the coarse structure whereas for the downscaling direction detailed feature information is lost. Given these notions, we find it difficult to give an explanation for our experimental results. Also we find it surprising that existing literature on feature approximations [13, 3, 72] does not study the effect of different approximation strategies with respect to approximation direction. It seems like we are the first to perform this experiment, with an interesting outcome that has led to a significant improvement in the detection rate of our final detector.

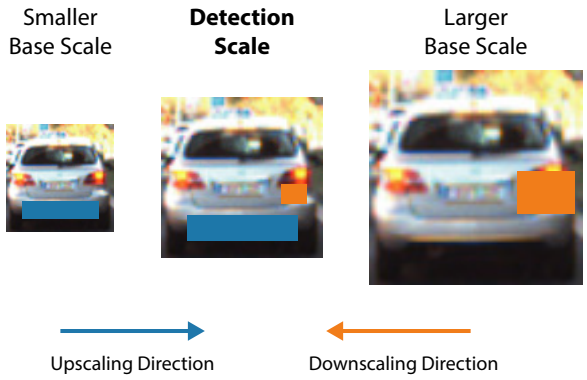


Figure 8.1: ***Feature Approximation Directions.*** For a current detection scale the nearest *smaller* or *larger* base scale can be selected as starting point for the approximation. Performing the approximation always in downscaling direction (orange) is found to be significantly better.

8.2 Future Work

Computational Improvements. From a computational point of view, we find it hard to think of additional techniques for speeding up the detector beyond marginal increases. Taking into account the number of calculations per image and the specifications of the current hardware, we believe the frame rates achieved by our detector are impressive running on a laptop. It is unlikely that channel extraction can be performed faster than the current implementation. Our highly optimized OpenCL filter kernels running on the GPU are maximally taking advantage of the number of processing units of a modern laptop. For hardware devices without shared memory between CPU and GPU a small speed gain can be achieved by streamlining the process of copying images between the two processing units. Our benchmarks on *integral* image computation shows that the bottleneck is the data bandwidth between the two processing units. Decreasing the number of image copy operations can therefore result in a minor speed improvement. In the final part of our future work we propose an improved detection pipeline with classification on the GPU. Given the high parallelism of our solution, it will directly benefit from future hardware improvements.

Image and Feature Normalization. We believe most false negatives (missed detections) of our detector are due to our relatively simple feature representation and limited resolution of evaluation images. Features are computed as summation over a certain region within the model size and the classification of decision stumps is performed simply by thresholding the channel feature response. This process can be executed very efficiently, mainly due to the availability of the integral images. However the process lacks the ability to perform normalization on intensity levels [94]. Analyzing the ground truth annotations on the TME dataset that are missed by our detector we observe that many of these occur in bright lighting conditions. Low-level details can have an important impact on the final detection rate. An example is the local and global normalization scheme applied to the input images. For our work we have not studied the effect of image normalization

on the final detection rate. For HOG features the importance of local normalization is emphasized in [10]. The survey paper [4] demonstrates the effect of various image normalization schemes in the context of integral channel features. The authors discover that global normalization of the input images is surprisingly effective in improving the detection rate. For pedestrian recognition the authors report an 8% increase in detection rate when using global normalization in comparison to no normalization. For future work on our own detector we suggest exploring the effect on detection rate and speed with the inclusion of global image normalization.

Occlusion Handling. One of the disadvantages accompanied with using a simple feature representation in boosting framework using only depth-1 decision trees is the limited capability of detecting *partially occluded* objects. Visual inspection of our detector running over videos indicate that objects that are more than 50% occluded are rarely marked as detection. Given our simple feature representation this is not surprising. One of the approaches for improving the detection rate of occluded objects is to train a set of occlusion-specific classifiers each with a specific occlusion-level. That is, there is number of occlusion-specific detectors corresponding to the each occlusion-direction. In the context of car detection, occlusions from the left and right are the most important. For some scenarios it might also be necessary to consider occlusions from the bottom. Studying pedestrian detection using channel features, Mathias *et al.* [56] propose an interesting technique for training multiple occlusion-specific detectors and reusing features at detection time. This is an compelling proof of concept that could serve as next improvement to our detector. For car detection in highway scenarios we can imagine that a number of assumptions on the road traffic dynamics/patterns can significantly reduce the amount of possible occlusion types to handle. Notice that with the availability of a depth-map at detection time, it is possible to determine the direction of the occlusion and use only the appropriate classifier reducing the number of model evaluations per subwindow.

Increasing Model Complexity. Improvements to detection quality are most likely to be achieved by improving the feature representation or closely related, the type of weak classifiers. Our boosting cascade classifies each subwindow by simple evaluation of decision stumps (depth-1 decision tree). For pedestrian detection it was shown [4, 16] that depth-2 decision trees can improve the detection rate with approximately 5%. Combining multiple features in higher-depth decision trees allows a more complex partitioning of the feature space therefore resulting in potentially better classification results. For depth-2 decision trees each weak classifier combines three channel features as illustrated in Figure 8.2. As a consequence the detection speeds will also suffer since each weak classifier evaluation requires extraction of three features instead of one. Increasing the complexity of our weak learners enables us to train more complex models, however we expect that this also requires a larger training dataset to reduce negative generalization effects due to overfitting.

Just in Time Channel Computation. The overall feature extraction and classification process is also already pushed to its limits with regard to computation efficiency. Our visualizations for WaldBoost displaying the number of features evaluated per window

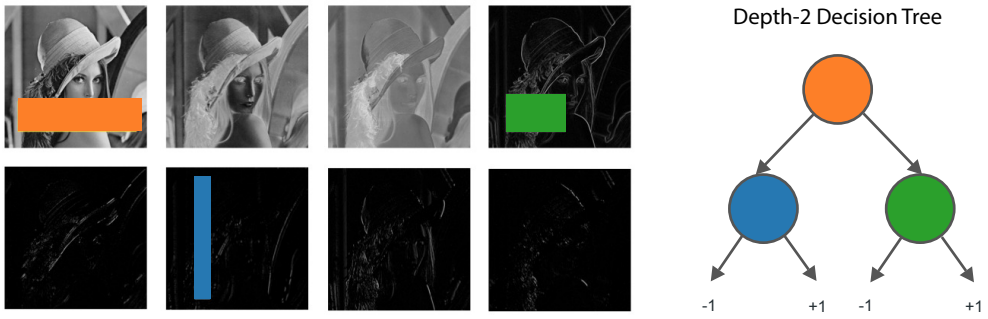


Figure 8.2: ***Depth-2 Decision Tree.*** More complex weak classifiers combine multiple features. In contrast to decision stumps, the depth-2 decision tree predicts the label based on three features from one or more channels. The increased complexity of the weak classifiers most likely improves recall rate.

show that not-promising subwindows are almost always rejected after evaluation of the first weak classifier. Therefore our detector spends almost no computational resources on processing regions that are not interesting to the detection task. An interesting idea for speeding up the detection pipeline is what we refer to as *just in time* computation of channels and their integral images. For non-promising image regions it seems cumbersome to compute all 10 image channels and their integral images if their information is never used prior to rejecting the entire region. Our experiments demonstrate the importance of the gradient orientation channel encoding horizontal edges. For car detection the first features to be computed in the classifier cascade are typically extracted from this orientation channel. Therefore for non-promising image regions, extracting the remaining 9 image channels is unnecessary and causes slowdown. Just in time computation of image channels at the moment they are actually needed is one of our proposals for speeding up the detector. The same arguments hold for caching the *integral* images. When evaluating only a single feature summing the pixels directly from the channel might be faster than precomputing the integral image. Computing the integral image is only worthwhile when evaluating a large amount of pixel summations.

Search Space Reduction. The summary of results in the beginning of this chapter and the reports by Beneson *et al.* [3] show that search space reduction is one of the promising approaches for decreasing computation time per video frame. With reduction of search space we refer to decreasing the number pixels and scales examined by the sliding-window. Beneson *et al.* [3] use stereo-vision to build a "stixel world" for reducing the search space dramatically. While they achieve very high detection rates by reducing the search space to 640×60 pixels $\cdot$ 10 scales their method relies on the availability of depth information from a stereo camera. We are more interested in exploring search space reduction using a monocular camera. For this we propose exploiting the WaldBoost heatmap data that is already available from our detector, as displayed in Figure 7.7. This information is available for "free" as it comes as byproduct of the detection process. A straightforward method for reducing the search space could be to construct a *probability of occurrence map*. This allows us not only to determine promising spatial locations, but also can give

us information on what scales objects are most likely to occur. Such maps do not need constant updating but can be initialized in the beginning and updated after fixed time intervals. While we believe this is a very promising strategy one should be cautious with introducing bias in the search space by utilizing the output of the WaldBoost detector.

Multi-Class Detection. For improving our current detection framework we would focus on improving the detection quality without sacrificing on speed. Also extending the current techniques to multi-class handling is of direct use in a practical applications. With the addition of multi-class handling there is a entire new set of problems and possible solutions for improving efficiency. For example, if our goal is to detect both traffic *signs* and traffic *lights* we expect that there is a strong spatial correlation between the two. We might expect both object types to appear in the same region of the image, *e.g.* at the same height. Such spatial correlations can be encoded with the use of *crosstalk cascades* [14]. The proposal is to exploit spatial correlations by tightly coupling detector evaluation of nearby windows. This is achieved by introducing two opposing mechanisms: detector excitation of promising neighbors and inhibits of inferior neighbor windows. While this method is originally suggested for one-class detections we believe it is interesting to generalize the idea to encode multi-class spatial correlations to speedup the detection process.

Performance on Embedded Device. Our detector was originally designed to be as efficient and fast as possible, having in mind that it could run on embedded devices. While we have demonstrated a very fast detector, unfortunately we were unable to run our detector on an embedded device within the course of this project. The specific embedded platform that is available at TomTom for running computer vision applications is the Jetson TK1 platform featuring the NVIDIA Tegra K1 processor. However the main reason that we were unable to run our detector on this platform is the lack of OpenCL support for the Tegra K1 and Linux for Tegra. This is unfortunate since originally we decided to use OpenCL in favor of CUDA for its excellent portability and absence of vendor lock-in. As future work we hope to see our detector running on the embedded platform and are very interested in performing speed benchmarks. With adequate performance we foresee interesting applications of running object detections on embedded devices. In addition to car detection, our methods can also be used for detecting traffic signs, localization of traffic lights and forward collision warnings.

8.2.1 Classification on the GPU

In this final part of the future work we discuss a proposal for performing classification on the GPU. We motivate this approach by observing that classification of subwindows can be done independently hence massive parallelization can yield significant speed-up. An additional advantage of performing classification on the GPU is that the detection pipeline is streamlined. For hardware platforms without shared-memory between CPU and GPU, the proposal has the advantage of not having to send image channels back and forth. Given that these channels are large memory objects and the observation that transferring large amounts of data between CPU $\leftrightarrow$ GPU is a potential bottleneck, we

expect significant speed-up depending on the number of processing units and clock speed of the GPU. The proposed pipeline below discusses WaldBoost classification and multiscale feature pyramids without explicitly rescaling the input image and recomputing channels. We assume that the integral channels are already computed and are stored in GPU memory.

Performing classification of subwindows on the device the WaldBoost classifiers are first uploaded to global GPU memory. The idea is to upload WaldBoost classifiers trained at 4 base scales to the GPU, and use these to detect objects at all scales by multiscale feature approximations as discussed in Chapter 6. WaldBoost classifiers characterized by the decision stumps and stage rejection thresholds are transferred to global GPU memory. We propose an architecture in which detection scales are processed one by one. The CPU prepares so-called *scale requests* and uploads these to the GPU. Each scale request is characterized by a list of subwindows that need to be classified and a list of decision stump thresholds modified for the current scale according to the power law for feature scaling (Equation (6.25)). Upon receiving a scale request the GPU executes a thread for each subwindow to perform classification. Each thread performs typical WaldBoost classification and can be terminated at any stage when the sample is discarded by rejection thresholds. Since feature values can be computed by efficient pixel lookups from the integral images and the simple evaluation of decision stumps, classification of each subwindow should be very fast. As soon as all GPU threads are finished, the classifications are reported back to the CPU by only downloading the classification scores and labels. Final post-processing steps for each video frame, such as non-maxima suppression are performed by the CPU after detection results for all scales are returned by the GPU. The complete proposed detection framework with classification of subwindows on the GPU is displayed in Figure 8.3.

8.3 Conclusion

The foremost contribution of this research is the development of a high-speed detector and studying its behavior and characteristics throughout the design process. We have designed, implemented and studied an object detection framework which is capable of running at 150 frames per second over a search space $640 \times 150 \cdot 25$ scales on a modern laptop. In the context of rear-view car detection we evaluate our detector on the TME Motorway Sequence and report recall rates of 0.75 at a precision of 0.9 – Our detection framework uses both algorithmic and computational speed-ups to achieve these impressive frame rates running on a laptop. We believe that it is unlikely to achieve significant improvements in speed beyond incremental speed-ups. However given the high parallelism of our work, the detector will directly benefit from future hardware improvements. As follow-up work we suggest focusing on improving detection rate without scarifying on speed. We believe the compact feature representation in a boosting framework allowing efficient features evaluation one by one, is the only possibility for achieving real-time performance. Below we present a summary of our core contributions.

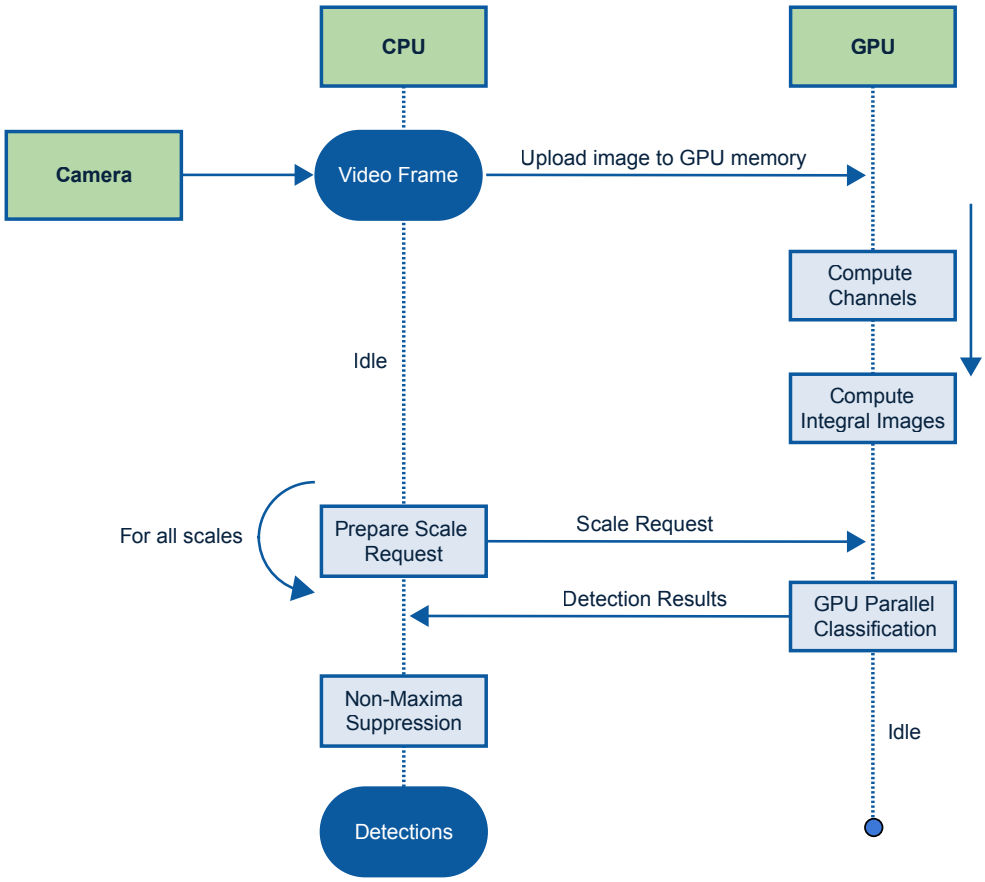


Figure 8.3: ***Proposed Framework with GPU Classification.*** Proposed detection framework with classification on the GPU. Note that integral images are not copied back to the CPU streamlining the overall detection process.

8.3.1 Summary of Contributions

- ***High-Speed Object Detector.*** The design of our detector and runtime benchmarks show that both a soft-cascade for speeding up classification of subwindows and better multiscale handling are necessary for achieving real time detection speeds. Without soft-cascade classification the detector spends too much time on feature extraction and evaluating weak classifiers. The early rejection of subwindows is crucial for reducing this time. Leaving out better multiscale handling in the form of feature approximations causes significant slowdown as a result of computing the image pyramid and integral channel computation for each scale. The combination of these techniques is demonstrated to work excellent together in a high-speed object detector. After putting all components together, the time spent on each component, *i.e.* channel computation, feature extraction and classification, is nicely balanced (Figure 7.22).

- ***Feature Representation for Car Detection.*** We have presented new insights in the feature representation for rear-view car detection. Vertically oriented gradients encoding horizontal edges are in particular important for the detection task. The combination of diverse, informative channels along with the integral image trick

for fast feature computation serves as a strong foundation for a high-speed object detection. Eventually, we hope to extend our results to different objects classes and improve occlusion handling.

- ***WaldBoost Detection.*** Motivated by the lowest number of weak classifiers evaluated in comparison to other soft-cascade approaches and the statistical validity we chose WaldBoost and have successfully demonstrated the effectiveness of the ratio test in an object detection framework. Our experiments show that on average only 2.1 weak classifiers are evaluated per subwindow for full images of highway scenes. There is only a small decrease of about 3% in detection rate caused by setting rejection thresholds too tight in comparison to Discrete AdaBoost. Visualizations demonstrate that our classifiers spends almost no computational resources on road, sky and trees in the image. One of the risks with WaldBoost is the difficult process of setting stage rejections thresholds by non-parametric density estimation on an independent validation set. As a result it is possible that the algorithm fails to to set rejection thresholds for the first stages which has the effect of slowing down overall the detection process.

- ***Multiscale Feature Approximations.*** We have demonstrated the usability of multiscale feature approximations based on the power law governing image statistics. Channel features aggregate pixel statistics over larger rectangular windows making them highly effective in the context of feature approximations. Our experiments show that the power law breaks down for smaller features. Additionally we have performed an interesting experiment on the approximation direction in the final detector. We report that performing approximations starting from the *larger* detector performs better than selecting the *nearest* base scale. Training models of multiple size has proven to be beneficial, however requires training examples to be available at high resolution. Visual inspection of our detector running over videos demonstrate seamless transitions from *exact* to *approximated* detections. There are many additional experiments to be explored for multiscale approximations and we believe it is an interesting strategy that can be used in a wide variety of object detectors. We expect many new object detectors adopting the feature approximation strategy in the years to come.

- ***TomTom Dataset.*** For this work we have constructed a novel dataset for rear-view car detection. Our semi-automatic strategy for compiling a dataset from mobile mapping camera data has proven useful for supervised learning. Our training data consists of 2000 rear-view car samples extracted from panorama images captured by TomTom mobile mapping vehicles. Uniformly at random we have sampled a large collection of panoramas captured by the fleet of mobile mapping cars within The Netherlands. From these panoramas all rear-view cars and vans are manually labeled and bounding boxes are extracted at size 80×80 to serve as training data for our detector.

8.4 Closing Remarks on Computer Vision

In this epilogue we consider some closing remarks with respect to our work and the computer vision research area in general. Since the code written for this project is not open source, we acknowledge that it is hard, if not impossible to *reproduce* the results reported in this thesis. We argue that this is one of the major problems with publications in computer vision and computer science in general. Getting algorithms to work requires significant amount of knowledge from both the theory, implementation and hardware. In particular computer vision algorithms seem to require various little tweaks and minor improvements to get the proposed idea working at the speed and quality that is claimed in a certain publication. Such details are almost never described in literature as they are not of interest to the typical reader. The difficulty of reproducing scientific claims as a result of ignoring such details, is one of the problems that the CV community is confronted with. It seems like forcing authors to publish their source code is the only way to solve this problem. However as the current research is a perfect example of, this is often not possible due to copyrights or legal issues.

While reading through a huge pile of CV papers in the early stage for this thesis, it strikes me that the majority of publications report incremental improvements to well-established methods. In the current publication model for computer vision it seems like papers with truly innovative ideas are more likely to get rejected at highly-selective conferences while papers describing minor advantages are more likely to pass the reviewers. Presumably the system and reviewers focus too much on short-term results and minor variations of existing methods. One of the underlying problems might be the enormous number of submissions that conferences such as CVPR, ICCV and NIPS receive. This causes the papers to be reviewed poorly hence more "obvious" papers, reporting minor improvements to an existing system, are more likely to pass the review in comparison to truly new proposals that require more careful analysis before acceptance.

8.5 Personal Reflection

For this thesis we focused on designing, implementing and analyzing a *complete* detection framework. As we have seen in this work, building the overall detector requires expertise from a wide variety of research areas, *i.e.* image processing, machine learning, multiscale handling and various computational speed-ups such as parallel computing techniques on both CPU and GPU. It is fair to say that the scope of this thesis is *broad*. Studying and implementing this broad set of topics is incredibly interesting, however the great number of different components to study, implement and optimize also makes it hard to really go in-depth within the course of a nine-month master thesis project. A project like this is never finished are there are many more experiments that I would have like to conduct but time did not allow me to.

Before starting this project I had never written C++ and was unfamiliar to OpenCV. After writing several thousands of line of code for this project, both of these have now become

second nature and are a valuable addition to my skill set. This is also true for GPU programming, writing a small Android application, exploring various C++ libraries, using TBB for parallelization and much more. Also this project presented me the perfect opportunity to use Python, NumPy and MatplotLib for data analysis and visualization. Finally the aspect that I almost forgot is that we spent the first month of this project on the topic of *contour tracking*, which we then discarded as a not so promising approach for detection and tracking of vehicles in on-road environments. Altogether this research was wonderful to conduct and has rewarded me with an incredible amount of knowledge, skills, insights and a first publication on my name.

Appendix A

Proofs and Derivations

A.1 Sequential Probability Ratio Test

For a sequential decision process the sample space is divided into three mutually exclusive parts:

$$S_t : (x_1, x_2, \ldots, x_t) \rightarrow \{-1, +1, \sharp\}. \tag{A.1}$$

For notational convenience we denote these three mutually exclusive regions in the sample space as $\mathcal{S}_t^0$, $\mathcal{S}_t^1$, $\mathcal{S}_t$ after t measurements. The statistical experiment is terminated at the smallest t for which the observation results in a decision $S_t \neq \sharp$. The *null hypothesis* H_0 is related to the classification as +1, whereas the *alternative hypothesis* H_1 corresponds to the classification as −1.

Suppose that before taking the first observation x_i there exists an *a priori* probability that H_0 is true. Denote the value of this *a priori* probability by $P(H_0)$. Likewise the probability $P(H_1) = 1 - P(H_0)$ denotes the *a priori* probability of H_1 being true. Not surprisingly due to the evidence obtained from the statistical experiment, the probabilities of H_0 and H_1 being true constantly change during the experiment. The *a posteriori* probabilities of the hypotheses being true after t observations are denoted by $P\left(H_0|x_1, \ldots, x_t\right)$ and $P\left(H_1|x_1, \ldots, x_t\right)$ respectively. Given the probabilities introduced above and using Bayes' rule we can write down the following relationships.

$$P(H_0 \mid x_1, \ldots, x_t) = \frac{P(H_0) \cdot P(x_1, \ldots, x_t \mid H_0)}{P(H_0)P(x_1, \ldots, x_t|H_0) + P(H_1)P(x_1, \ldots, x_t|H_1)} \tag{A.2}$$

$$P(H_1 \mid x_1, \ldots, x_t) = \frac{P(H_1) \cdot P(x_1, \ldots, x_t \mid H_1)}{P(H_0)P(x_1, \ldots, x_t \mid H_0) + P(H_1)P(x_1, \ldots, x_t \mid H_1)}. \tag{A.3}$$

In the following discussion, let P_{it} denote the probabilities $P(x_1, \ldots, x_t \mid H_i)$ calculated under the hypotheses $H_i,\ i \in \{0, 1\}$. Suppose that we want to construct a sequential test such that the conditional probability of a correct decision under condition that H_0 is accepted, is greater than or equal to d_0, and likewise for H_1 with d_1. For both probabilities

we make the restriction $d_0 > \frac{1}{2}$ and $d_1 > \frac{1}{2}$ because otherwise it might happen that the hypothesis with the smaller *a posteriori* probability will be accepted. Then the following sequential process seems reasonable [90]. For each observation, make a decision based on the observations to far:

$$S_t = \begin{cases} \text{Accept } H_0 \rightarrow \mathcal{S}_t^0 & \text{if } P\left(H_0 \mid x_1, \ldots, x_t\right) \geq d_0 \\ \text{Accept } H_1 \rightarrow \mathcal{S}_t^1 & \text{if } P\left(H_1 \mid x_1, \ldots, x_t\right) \geq d_1 \\ \sharp & \text{otherwise} \end{cases} \tag{A.4}$$

In order to be a valid experiment it is required that the sets $\mathcal{S}_t^0$, $\mathcal{S}_t^1$ and $\mathcal{S}_t$ are mutually exclusive and exhaustive. It can be shown [90] that this is indeed the case, and therefore that (A.4) is valid. To show this, first rewrite the inequalities (A.4) using Bayes' rule:

$$P\left(H_0 \mid x_1, \ldots, x_t\right) = \frac{P(H_0) \cdot P_{0t}}{P(H_0)P_{0t} + P(H_1)P_{1t}} \geq d_0 \tag{A.5}$$

$$P\left(H_1 \mid x_1, \ldots, x_t\right) = \frac{p(H_1) \cdot P_{1t}}{P(H_0)P_{0t} + P(H_1)P_{1t}} \geq d_1. \tag{A.6}$$

To show that these relationships are incompatible we derive a contradiction. First, by summing the inequalities we obtain:

$$P\left(H_0 \mid x_1, \ldots, x_t\right) + P\left(H_1 \mid x_1, \ldots, x_t\right) \geq d_0 + d_1 \tag{A.7}$$

Since $P\left(H_0 \mid x_1, \ldots, x_t\right) + P\left(H_1 \mid x_1, \ldots, x_t\right) = 1$, we have $1 \geq d_0 + d_1$. However considering that we have made the assumption $d_0, d_1 > \frac{1}{2}$ this is impossible. Therefore we have derived a contradiction and the proof is given that the sets $\mathcal{S}_t^0$, $\mathcal{S}_t^1$ and $\mathcal{S}_t$ are mutually exclusive and exhaustive.

Next, our goal is to find a relationship between the introduced constants d_0, d_1 and the probabilities and requirements for the statistical test. We begin by arranging Equations (A.5) and (A.6) in the following way.

$$\frac{P_{1t}}{P_{0t}} \geq \frac{P\left(H_0 \mid x_1, \ldots, x_t\right)}{P\left(H_1 \mid x_1, \ldots, x_t\right)} \cdot \frac{d_1}{1 - d_1} \tag{A.8}$$

$$\frac{P_{1t}}{P_{0t}} \leq \frac{P\left(H_0 \mid x_1, \ldots, x_t\right)}{P\left(H_1 \mid x_1, \ldots, x_t\right)} \cdot \frac{1 - d_0}{d_0} \tag{A.9}$$

This is an interesting result, since if the probability of H_0 being true is unknown, then these inequalities suggest the use of the following statistical test for making a decision for each observation t. At each stage, compute the **likelihood ratio** R_t given on the left hand side:

$$R_t = \frac{P_{1t}}{P_{0t}} = \frac{P(x_1, \ldots, x_t \mid H_1)}{P(x_1, \ldots, x_t \mid H_0)}. \tag{A.10}$$

To cover the exception of the special case $P_{1t} = P_{0t} = 0$, define $R_t = 1$ if this happens. Based on this ratio Wald [90] proposes the following statistical test to make a decision S_t:

$$S_t = \begin{cases} \text{Accept } H_0 & \text{if } R_t \leq B \\ \text{Accept } H_1 & \text{if } R_t \geq A \\ \sharp & \text{if } B < R_t < A \end{cases} \tag{A.11}$$

The constants A and B are chosen so that $0 < B < A$ and the sequential test has the desired values α and β characterizing the false negative and false positive rates respectively. Wald refers to the test given in (A.4) as the **Sequential Probability Ratio Test (SPRT)**. Although the derivation of the SPRT starts by mere intuition, it can be shown [90] that this test given an optimal solution if the following bounds A and B are adopted. The bounds A and B are directly related to the false positive and false negative error rates α and β. Following a proof involving cylindric points the following relationships can be derived:

$$\frac{\alpha}{1-\beta} \leq \frac{1}{A}, \qquad \frac{\beta}{1-\alpha} \leq B. \tag{A.12}$$

For details on this derivation we refer to Wald [90] page 128. These inequalities are very useful in practice for setting A and B if requirements on error rates are specified. For practical applications the Wald suggests setting A and B according to these bounds.

A.2 Power Law Derivation

Let $\phi(I)$ denote an arbitrary (scaler) image statistic and $E[\cdot]$ an expectation value of an ensemble of natural images. Ruderman and Bialek [71, 69] made the fundamental discovery that the ratio $E[\phi(I_{s_1})]$ to $E[\phi(I_{s_2})]$ computed over two image ensembles of natural images at scales s_1 and s_2 respectively, depends only on the ratio s_1/s_2 and is independent of the absolute scales. In this section we show that these findings imply that $E[\phi(I_s)]$ follows a power law:

$$\frac{E[\phi(I_{s_1})]}{E[\phi(I_{s_2})]} = \left(\frac{s_1}{s_2}\right)^{-\lambda_\phi} \tag{A.13}$$

Proof. Assume that the empirical observation by Ruderman and Bialek holds, *i.e.* the ratio between the two expectation values is a function of the relative scale difference s_1/s_2. Denote $\mathcal{R}(s) = E[\phi(I_s)]$ for notational convenience. Then we can introduce a function f which expresses the observation of Ruderman and Bialek:

$$\frac{\mathcal{R}(s_1)}{\mathcal{R}(s_2)} = f(s_1/s_2). \tag{A.14}$$

We can apply this relationship iteratively until we reach $s_1 = 1$ or $s_2 = 1$. Arriving at this "end scales" we obtain we following two relationships.

$$\frac{\mathcal{R}(s_1)}{\mathcal{R}(1)} = f(s_1) \tag{A.15}$$

$$\frac{\mathcal{R}(1)}{\mathcal{R}(s_2)} = f(1/s_2) \tag{A.16}$$

From this we can derive a requirement on function f by observing that after rearrangement:

$$f(s_1/s_2) = f(s_1)f(1/s_2). \tag{A.17}$$

Given this observation, introduce a change of variables $f'(s) = f(e^s)$. and recognize that given this introduction the following must hold.

$$f'(s_1 + s_2) = f'(s_1)f'(s_2). \tag{A.18}$$

If additionally f' is a complex, continuous function of real variables, then it can be shown [37] that either $f'(s) = 0$ or there exists a complex number α such that $f'(s)$ can be written in the form:

$$f'(s) = e^{-\alpha s}, \quad \forall s \in \mathbb{R} \tag{A.19}$$

Hover since our change of variables, $f'(s) = f(e^s)$ we can combine (A.19) to obtain:

$$f(s) = f'(\ln(s)) = e^{-\alpha \ln(s)} = s^{-\alpha}. \tag{A.20}$$

Where we have used that $-\lambda \ln(s) = \ln(s^{-\lambda})$. With this we have proven that $f(s)$ follows a power law directly corresponding to relationship (A.13).

□

Appendix B

Conference Paper

In this final appendix we include the scientific paper that was written based on this thesis together with Robert Lukassen, Marco Loog and Lu Zhang. The paper is accepted as (poster) presentation to **The Netherlands Conference on Computer Vision (NCCV)** that will be held on September 14-15, 2015 in Lunteren, The Netherlands. In this paper we focus on the system design of our object detector and give a brief introduction of all components contributing to the high detection speeds. We emphasize that this paper can be not be considered a summary of the current thesis, since a number of important topics, theory and experiments are not described in the paper due to the page limit of eight pages.

The System Design of a High-Speed Object Detector

Tom Runia
Delft University of Technology*
Pattern Recognition Laboratory
tomrunia@gmail.com

Robert Lukassen
TomTom
Product Lab, Eindhoven
robert.lukassen@tomtom.com

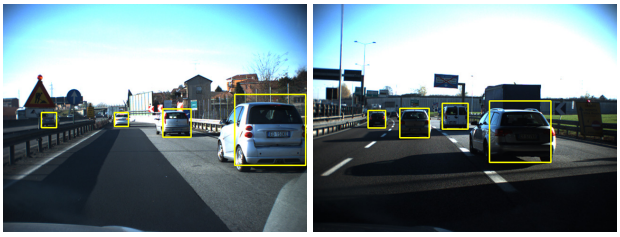
Lu Zhang Marco Loog
Delft University of Technology
Pattern Recognition Laboratory
{lu.zhang,m.loog}@tudelft.nl

Abstract

We discuss the design of a high-speed object detection framework. More specifically we focus on rear view car detection. However the methods discussed are not limited to this object class. Our baseline detector uses integral channel features in a boosting framework reminiscent of Viola-Jones. With our baseline detector as starting point we consider the implementation and study of two approximation techniques that result in a significant detection speedup. The first approximation focuses on decreasing the time spent on classification of subwindows. The second approximation technique targets the multiscale aspect of our object detector. We show that these techniques result in a $17.5\times$ *speedup without sacrificing on detection rate. Based on these approximation techniques and a fast GPU implementation for extracting channel features we report detection speeds up to* 55 *fps without exploiting scene geometry or reducing the search space.*

1. Introduction

The automotive and navigation industry shows increasing interest in the development of computer vision systems [5, 13]. More and more cars are equipped with cameras that can sense the world around the vehicle for example by capturing video and running complex computer vision algorithms to analyze the data. In the context of *Advanced Driver Assistance Systems* (ADAS) computer vision can prevent collisions or fatal injuries by forward collision warnings [17], pedestrian recognition [6] or car detection [5]. The process of automatic map making can benefit from aggregating computer vision observations from cars driving around. Vision-based localization of traffic signs, traffic lights, road closings, accidents and parking places can be of great value for drivers and advanced driver assistance. While the detection of static objects such as traffic signs is closely related, in the current work we focus on the detection of other vehicles on the road. More precisely we focus on highway scenarios in which *rear view car detection* (Figure 1) is one of the important problems in computer vision for highly automated driving.



(a) Occlusions **(b)** Difficult lighting

Figure 1: Frames from the TME Motorway Sequence [5]. Only cars and vans are labeled; annotations are rescaled to square size.

In this work we offer a look behind the scenes for designing a high-speed object detector. Based on an exhaustive review of existing literature on time constrained object detection we select methods that find the right balance between speed and detection rate. The detection framework we propose and implement builds upon the work on integral channel features [8], WaldBoost classification [22] and publications on feature scaling approximations for fast multiscale detection [7, 1]. The core contribution is the integration of these high-performance vision algorithms into a full detection framework and the analysis of the contributions of these techniques to high-speed performance. In other words, we show what algorithms yield the most important contributions to achieving real-time detection speeds and potential sacrifices in detection accuracy. We also discuss the technical aspect of our detection framework by giving a number of implementation details.

Similar to many modern object detectors, our work utilizes rigid templates trained at a certain size (*e.g.* 64×64) that at detection time are applied in a sliding window fashion to full images. As mentioned, we focus on the development of a rear view car detector, however we emphasize that all methods could easily be extended to other object classes.

*This research was conducted at TomTom Product Lab in Eindhoven.

Regarding the vision system, we opted for a monocular camera, mounted for example on the dashboard or just below the rear view mirror. In this work we exclude the components of tracking, depth-estimation, temporal information, geometric scene information and additional prior knowledge. Each image is analyzed "as is" meaning that no information from preceding frames is used during the detection process.

The paper is organized as follows. In Section 2 we discuss a number of recent contributions on high-speed object detection that are related to our work. Section 3 introduces our feature presentation, learning framework and baseline detector. This is followed by two sections focusing on algorithmic speedups for our baseline detector: Section 4 describes WaldBoost for speeding up the classification, while Section 5 introduces the idea of feature rescaling for fast multiscale detection. Our complete detection framework, hardware setup, implementation and experimental results are presented in Section 6. In Section 7 we discuss and summarize our results.

2. Related Work

Two object detector types that are known for its excellent detection rates are the works by Felzenswalb *et al*. [10] on deformable part based models (DPM) featuring pictorial structures, and recent advances on convolutional neural networks [14]. However these methods tend to focus more on achieving the highest detection accuracy while neglecting the need for speed. For example we report processing times of more than 2 seconds per 1024×768 image using the DPM detector on a modern laptop. For the current work we are interested in running object detection in real-time on embedded devices, hence we do not consider these two detection frameworks.

Providing an exhaustive overview of publications on high-speed object detection is beyond the scope of this paper. However we will summarize a number of techniques that are related to our work and we believe are important to achieving the highest possible detection speeds. Speed improvements for existing object detection frameworks can roughly be categorized into (1) improving the feature representation, (2) speeding-up the classification of subwindows, (3) focusing on fast multiscale detections and (4)computational improvements. Below we summarize important contributions to these topics in relation to sliding-window based object detector.

Feature Representation. Over the last couple of years there tends to be more focus on improving features rather than on improving classification [3]. Meaningful and efficiently extractable features are crucial to good detection rates. The first generation of object detectors frequently employed Haar features. While fast to compute using integral images, the popularity of Haar features decreased mainly due to the introduction of histograms of oriented gradient (HOG) features. A recent survey paper on pedestrian detection [2] shows that many of the high-performing detectors use a HOG-like feature representation. It seems like gradient orientation-based features are highly efficient in encoding edge, corner, key-point and local-shape information, while being reasonably fast to extract. Although HOG features with SVM classification [6] are not particularly slow, the specific normalization scheme and high dimensional feature vectors hinder computational speedups. For our detector we use integral channel features [8], which are versatile, information rich and efficient to compute.

Classification Speedups. Inspired by the cascade structure of classifiers in increasing complexity by Viola and Jones, a number of papers appeared on so-called *soft-cascades* [22, 28, 27] for speeding up boosted classification. These works focus on pruning negative subwindows as early as possible in the detection process. During the training process in addition to selecting the best features and thresholds also rejection thresholds are learned. When at evaluation time the cascade score drops below this threshold the subwindow is immediately pruned and labeled as non-object window. Typically non-promising windows are rejected early in the cascade resulting in a significant speedup when the number of windows to evaluate is large as with full image detections. We implement and study WaldBoost [22] for speeding up the classification of subwindows.

Multiscale Models. Typically for detecting objects of various size, the input images are rescaled a number of times (*e.g*. 30 scales) and features are extracted for each scale. Computing the image pyramid and extracting features at every scale requires a significant amount of computational resources. To overcome this computational burden some authors propose interpolation of features over scales to reduce the number of scales in the image pyramid. Dollár *et al*. [7] introduce a method for approximating various feature types over adjacent scales using the so-called power law for feature scaling. Since then this technique has been employed with great success for pedestrian detection [1] and general object detection using deformable part models by Saghedi *et al*. [20]. This technique is also an important aspect of our detector and dramatically reduces computation time spent on image rescaling and feature extraction (see Section 5).

Computational Speedups. In a different line of work, several improvements focusing on computational speedups have been proposed. Zhu *et al*. [29] show that HOG features can be rapidly computed using integral images. Furthermore various GPU implementations for speeding

up computer vision algorithms are described [18], and a number of low-level operations are optimized with vector operations, multiple cores and CPU cache management [20].

The work by Beneson *et al*. [1] shares the most overlap with our work. Similar to our work the authors propose a very fast object detector for pedestrian detection using integral channel features, boosting and an approximation scheme for multiscale detections. Core component for reaching high detection speed in their detection framework is the ground plane estimation technique for reducing the search space. The authors report detection speeds of 100 fps, using the ground plane estimation technique and stereo-camera. Using the stereo-camera setup and the ground plane estimation technique the authors are able to reduce the search space to 640×60 pixels regions over 10 scales, hence this allows them to run the detector at such high speeds. Our work is different since we do not consider ground plane estimation but rather focus on speeding-up the classification of subwindows.

3. Baseline Detector

Rear view car detection is a classic instance of rigid object detection. We train an object detector of fixed size and perform detection in full images using the *sliding window* methodology. Our detection framework is based on *integral channel features* [8], which combine the information richness of HOG features with the computational efficiency of Haar features. We motivate this decision by observing that – with a proper set of channels – the work on channel features is closely related to HOG features and at the same time has been shown to outperform other features computationally [8, 16].

3.1. Integral Channel Features

The general idea behind integral channel features is that multiple registered image channels $C(i, j)$ are computed using shift-invariant transformations Ω applied to an input image, *i.e*. $C(i, j) = \Omega[I](i, j)$. First-order integral channel features f_Ω are defined as the sum of pixels in a fixed rectangular region R in a single image channel C:

$$f_\Omega(I) = \sum_{i,j \in R} C(i, j) \tag{1}$$

Using integral images for each channel such features are extremely fast to compute. Experiments in [8] show that the combination of color + gradient magnitude + six gradient orientation channels (Figure 2) typically result in the best detection performance. The LUV color space is superior compared to other color spaces and the addition of more than six gradient orientation channels shows no improvements in detection rate for pedestrian detection [8].

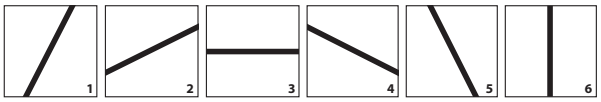


Figure 2: Gradient orientations ("unsigned" versions are utilized)

3.2. Learning Framework

The training process of our detector is similar to Viola-Jones [25] and the work by Dollár *et al*. [8]. Rather than carefully designing features we generate a large feature pool of random features and adopt **Discrete AdaBoost** [11] for greedy feature selection. Each feature is described by a randomly generated rectangle with minimum area of 25 pixels and a randomly selected channel index 0–9. In each training stage t the best *weak learner* h_j restricted to a single feature j is trained by optimizing the decision threshold and polarity. The strong classifier H_T returned by the training process is an ordered sequence of all T weak learners in which the associated weight α_t assigned to each weak learner is inversely proportional to the training error of that stage. Deciding upon the label $y = \{-1, +1\}$ of an object x is done by thresholding the cascade score $H_t(x)$ with is linear combination of the weak classifiers:

$$H_t(x) = \sum_{t=1 \ldots T} \alpha_t \cdot h_t(x) \tag{2}$$

For all experiments reported in this paper we set $T = 400$ (feature dimensionality). For 640×480 images at 25 scales our baseline detector spends approximately 4.5% of the processing time on image rescaling and channel extraction, 32% on feature extraction and 63% on classification of subwindows. However the actual contribution to the total processing time per image is different since feature extraction and classification run in a parallelized loop.

4. Soft-Cascade Classification

Cascade detections have been shown to operate extremely fast making them a good choice for high-speed object detection. Fast rejection of non-promising subwindows is crucial to high detection speeds, motivated by the observation that the majority of tested subwindows does not contain the object of interest. Since the original introduction of the cascade structure by Viola-Jones [25] many publications have been devoted to improving cascade learning and detection. One particular problem of the original cascade structure is determining the optimal target detection rates for each stage of the cascade. This is one of the problems addressed by *soft-cascades*.

The structure of soft-cascades is different to the cascade proposed by Viola-Jones since only one fine-grained classifier of T weak learners is constructed. Various soft-cascade

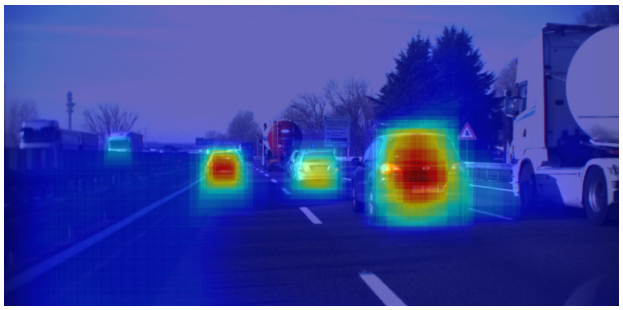


Figure 3: WaldBoost attention heatmap. For each pixel in the images we visualize the number of weak classifiers that was evaluated. For blue regions the subwindows were rejected almost immediately, while for red regions the entire cascade was evaluated.[1]

approaches [22, 28, 27, 4] haven been described, each focusing on learning stage thresholds $\theta(t), t \in \{1,\dots,T-1\}$ for rejecting non-promising subwindows at the earliest stage possible. Zhang and Viola [28] formulate the process of learning stage thresholds as multiple instance learning problem (MIP). This approach is successfully adopted in detection frameworks [8, 7] due to its simplicity and effectiveness. For our work we choose to adopt WaldBoost as soft-cascade approach to speedup classification. Experiments [28] have demonstrated that WaldBoost is faster than MIP for rejecting subwindows, and we appreciate the statistical foundation of WaldBoost, *i.e.* the sequential probability ratio test used for calculating stage thresholds is proven optimal, while the other methods are more designed based on intuition and empirical observations.

4.1. WaldBoost

WaldBoost is an extension of AdaBoost and formulates the classification problem as a *sequential decision process*. Based on the current cascade score H_t (*trace*) the sequential decision process S_t offers three possible options:

$$S_t : (h_1, h_2, \dots, h_t) \rightarrow \{-1, +1, \sharp\} \tag{3}$$

Where $\sharp$ means 'continue' when the object is undecided from the first t measurements. The sequential decision strategy is formulated to minimize the average decision time $\overline{T}_S = E[T_S(x)]$, where the decision time $T_s(x)$ denotes the minimal t for which $S_t \neq \sharp$. To meet system requirements, the user specifies the allowed false negative rate α (*e.g.* $0.5 \cdot 10^{-4}$) and allowed false positive rate β (typically set to 0 for object detection). Wald [26] has proved that the optimal strategy for this decision problem can be expressed

[1]For a video visualization of the WaldBoost attention heatmap constructed for this article, see: `https://www.youtube.com/watch?v=4OlITWa1bTY`

Table 1: Comparison in the number of weak classifiers evaluated for AdaBoost and WaldBoost per subwindow. For AdaBoost the label for each subwindow is determined after processing all $T = 400$ stages. On average WaldBoost only requires 2.1 weak classifier evaluations per subwindow. Statistics are computed over 200 images 1024×768.

	AdaBoost	**WaldBoost**
Windows per image	$2.3 \cdot 10^5$	$2.3 \cdot 10^5$
Tot. features extracted	$9.4 \cdot 10^7$	$4.8 \cdot 10^5$
Avg. stages per window, $\overline{T}_S$	400	**2.08**

in terms of the likelihood ratio $R_t(x)$:

$$R_t(x) = \frac{P(h_1(x),\dots,h_t(x) \mid y=-1)}{P(h_1(x),\dots,h_t(x) \mid y=+1)} \tag{4}$$

Given the likelihood ratio $R_t(x)$ and constraints on the error-rates, Wald proposes the *sequential probability ratio test* (SPRT) for setting rejection thresholds $\theta_A^{(t)}, \theta_B^{(t)}$ in each stage of the decision process. To avoid the computation of $R_t(x)$ involving a high dimensional density estimation, Šochman and Matas [22] suggest projection of the t-dimensional space into a one-dimensional space using the strong classifier score up until the current stage (Equation (2)). This simplifies the density estimation of $P(h_1(x),\dots,h_t(x) \mid y=\pm 1)$ in Equation (4) to a one-dimensional density estimate for which Parzen windows are adopted. After estimating the probability densities, stage thresholds can be determined using α, β. For all experiments in this paper we set $\alpha = 0.005$ and $\beta = 0$, *i.e.* we allow 0.5% false positives and no false negatives. See Figure 4 for an illustration of the stage threshold estimation process. During the detection phase, stage thresholds $\theta_A^{(t)}, \theta_B^{(t)}$ are applied for rejecting subwindows as early as possible:

$$S_t^* = \begin{cases} +1 & \text{if } H_t(x) \geq \theta_B^{(t)} \\ -1 & \text{if } H_t(x) \leq \theta_A^{(t)} \\ \sharp & \text{if } \theta_A^{(t)} < H_t(x) < \theta_B^{(t)} \end{cases} \tag{5}$$

Using this strategy most of non-promising subwindows are rejected after evaluating only one or two stages. The visualization in Figure 3 illustrates this effect, while Table 1 shows significant reduction in the average number of weak classifiers $\overline{T}_S$ evaluated in comparison to AdaBoost. Extending our baseline detector with WaldBoost decreases the time spent on feature extraction to 22%, classification to 40% and consequently increases the fraction of time spent on channel computation to 38%.

5. Fast Multiscale Detections

Conventional object detectors construct an image pyramid to detect objects with variable sizes. The computation

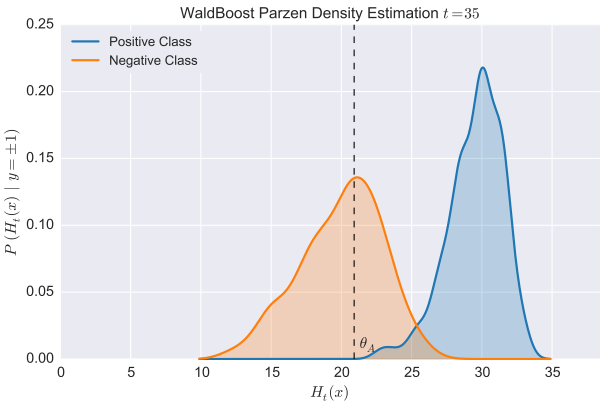


Figure 4: Parzen density estimate of $H_t(x)$ distribution computed on the validation dataset (500 examples). Wald's SPRT is used for computing stage threshold θ_A given the user-specified α.

of the image pyramid and feature computation for each scale typically is the bottleneck in modern object detectors. Based on important works on scale space [15] and experiments on scale behavior of image statistics [24], Dollár *et al*. [7] propose an elegant technique for multiscale feature approximation to speedup multiscale detections.

Early studies [19, 23] on image statistics show that natural image spectra follow a power law. In context of object detection using integral channel features, let I be an image available at scales s_1 and s_2 denoted by I_{s_1} and I_{s_2} and a channel feature f_Ω. The power law describing scale behavior of image statistics over an ensemble is the foundation of the work by Dollár *et al*. describing the power law for feature scaling which expresses that the ratio between features at two different scales is only a function of the relative scale difference:

$$\frac{f_\Omega(I_{s_1})}{f_\Omega(I_{s_2})} = \left(\frac{s_1}{s_2}\right)^{-\lambda_\Omega} + \varepsilon \tag{6}$$

Where ε indicates the deviation from the power law and λ_Ω is the power law parameter characteristic for a certain channel Ω. Originally the validity of this power law was found to hold for *ensembles* of images, however it was shown by [7] that this also holds for individual images by decomposing the image into an ensemble of patches (small images). For fast multiscale detections Dollár *et al*. [7] rescale the input image two times per octave (*base scales*) and exploit the feature approximation relationship for interpolation to nearby scales. This significantly reduces the number of rescalings and time spent on feature extraction. For each channel λ_Ω was computed using the methodology as explained in [7].

5.1. Object Detection without Image Resizing

Beneson *et al*. [1] built upon the multiscale feature approximation techniques by proposing object detection without image resizing. Their core idea is to move resizing of the image from test time to training time. Similar to their detection framework we train our models at four different scales, *i.e*. $s = 0.5, 1, 2, 4$ – While this is not practically feasible for many object detectors due to excessive training times, our detector trains in approximately 40 minutes. Consequently training the detector at four different scales can be done in 2.5 hours on a fast desktop using a parallel implementation. At runtime we use the four models at different scales as base scales and apply the power law for feature scaling (6) to approximate intermediate scales.

One problem that arises when training the detector at multiple scales is the availability of training examples in high resolution. In our current setting the training examples are available at 80×80, but our largest detector model is trained at 256×256. This unavoidably causes negative upscaling effects certainly resulting in a decrease in detection performance. In the ideal case, training examples at all scales should be available although we recognize the difficulty of putting such dataset together.

6. Detection Framework and Experiments

Bringing together all the parts discussed in the previous sections we put together our object detector. The architecture of our full detection framework is displayed in Figure 5. Upon receiving a video frame from the camera the image is sent to GPU memory. OpenCL filter kernels compute the 10 registered image channels and send these back to CPU memory. Surprisingly enough computing the *integral* channels on the GPU and then pushing these back to CPU was found to be slower than our current implementation. This can be explained by the fact that 10 integral channels are quite large memory objects hence the data bandwidth between CPU and GPU becomes the bottleneck. Once computation of the integral channels is finished, the classification of all subwindows over all scales is started in parallel. We emphasize that *no* image rescaling is performed but rather we use the models trained at 4 different scales and use these 'base' scales for our approximations to other scales. After processing all scales, nearby detections are combined using a simple non-maxima suppression method described in [8].

Hardware. Experiments for this paper are performed on a MacBook Pro (late 2013) running OS X 10.9 – This laptop contains a dual-core Intel i5-4258U processor running at 2.4GHz and is supported by an integrated Intel Iris 3000 GPU running at 1.2GHz over 280 processing units. The detector proposed in this work was designed to run on an

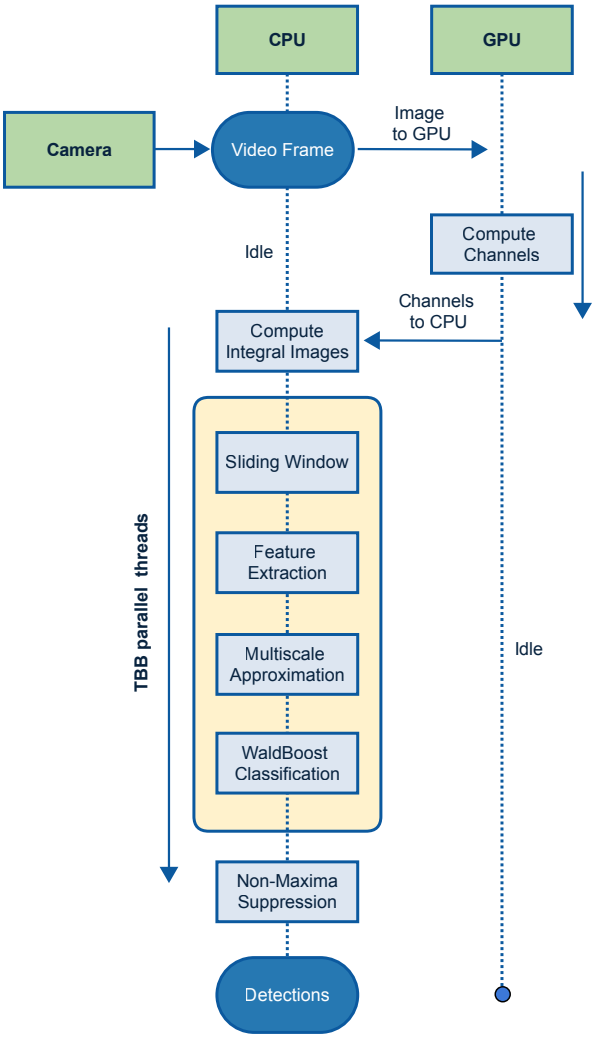


Figure 5: Architecture of our object detection framework. In our current implementation the GPU only applies filter kernels for computing the 10 image channels. Feature extraction, multiscale approximations and classification are performed on the CPU in a parallel implementation.

embedded platform, although unfortunately we have not yet performed experiments on an embedded device. Our specific embedded platform is the Jetson TK1 platform featuring the NVIDIA Tegra K1 processor. However NVIDIA has not released OpenCL support for the Tegra K1 platform (Linux for Tegra) making it impossible to run our algorithms relying on OpenCL.

Implementation. Our detector is written from scratch in C++11 building on top of OpenCV 3.0.0, although we only use its core functionality.

Figure 6: Top: positive training samples, in total 2000. **Bottom:** randomly selected negative training examples not containing cars and vans, in total 4000.

Intel's Task Building Blocks (TBB) was selected for CPU parallelization. For GPU parallelization we choose OpenCL in favor of CUDA. The flexibility and portability of OpenCL is more important to us than the minor speed increase CUDA has to offer at the cost of vendor lock-in.

Image Channels. For extraction of our channel images we utilize fast OpenCL kernels to run image processing operations. In our implementation we use the unsigned gradient orientations displayed in Figure 2, extracted using simple $[-1,0,+1]$ and $[-1,0,+1]^T$ filter kernels without prior smoothing [6]. The cube root $\sqrt[3]{x}$ calculations required for LUV color conversion are speedup using Halley's method [12] for providing a fast approximation with small error. The gradient orientation computation $\Theta = \text{atan2}(G_y, G_x)$ using outputs G_x, G_y from the derivative masks is speedup using an approximation for the inverse tangent described in [21]. The extraction of all 10 channels using our GPU implementation and downloading the channels back to CPU memory runs at 40 fps for 1024×768 images and 92 fps for 640×480 images on our hardware. Note that this is almost five times as fast as the original implementation [8] reporting channel extraction running at 40 fps on 320×240 images.

Training Data. Our training data consists of 2000 rear view car samples extracted from panorama images captured by TomTom mobile mapping (MoMa) vehicles. Uniformly at random we have sampled a large collection of panoramas captured by the fleet of MoMa cars within The Netherlands. From these panoramas all rear view cars and vans are manually labeled and bounding boxes are extracted at size 80×80 to serve as training data for our detector. Additionally 4000 random patches, not containing cars or vans, are extracted as negative training data. Figure 6 displays a subset our training examples.

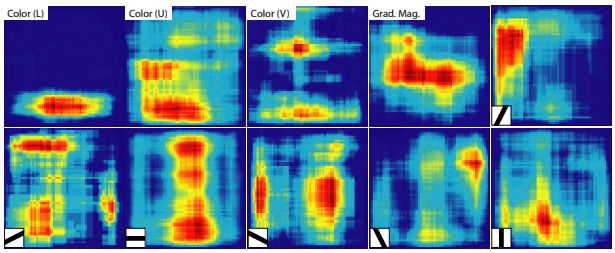


Figure 7: Feature selection by AdaBoost for rear view car detection. The gradient orientation channel corresponding to horizontal lines accounts for 25% of all features, far more than the other channels. These gradients have the strongest response around the license plate and near the roof of the car. The first color channel focuses on the dark shadow on the ground just below the car.

Test Data. For the evaluation of our object detector we report detection results for the TME Motorway Sequence [5]. The videos are captured by a dashboard mounted camera while driving highways in Northern Italy at 1024×768 (20 Hz). The total of 9 daylight sequences include various traffic situations, number of lanes, road curvature and lighting conditions. Two example images are displayed in Figure 1. For evaluation we computed statistics over 465 video frames each time with an interval of 50 intermediate frames. Cars and vans with more than 50% occlusion are excluded as ground truth.

Evaluation Metrics. The PASCAL VOC overlap criterion [9] is used for matching detections to ground truths and for determining the number of true positives (TP), false positives (FP) and false negatives (FN). To be considered a correct detection, the overlap ratio between the predicted bounding box and the ground truth bounding box must exceed 50% according to PASCAL VOC definition [9].

Training Time. Training of our baseline detector and learning stage thresholds using WaldBoost is fast in comparison to other detectors: training the detector at a single scale with 600 weak classifiers and an initial feature pool of 40.000 features takes about 40 minutes on a powerful desktop (Intel Core i7-2600). Note that the training process runs completely on the CPU and is parallelized using TBB. Prior to the training process all feature values for all training samples are precomputed and loaded into RAM, resulting in fast training stages. Training our multiscale detector on 4 scales finishes in 2.5 hours.

Learned Features. An interesting observation made from analyzing the features learned for rear view car detection is the fact that over 25% of the channel features correspond to vertically oriented directional derivatives, *i.e.* they correspond to horizontal edges (see Figure 2). Remaining color and gradient magnitude and orientation channels each account for 5–12% of the total number of features.

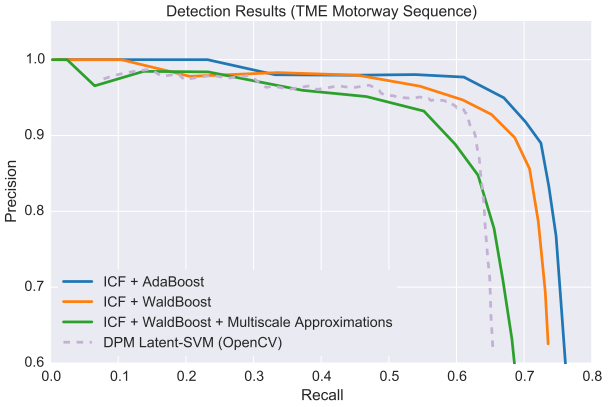


Figure 8: Detection quality comparison on TME Motorway Sequence. As comparison we show the quality of rear view car detections for the Felzenswalb detector [10] (OpenCV implementation).

Table 2: Speed benchmark over 640×480 images analyzed at 30 scales. Runtimes are averaged over 200 video frames; for all settings the strong classifier consists of 400 weak learners.

	Relative Speed	**Absolute Speed**
Baseline (ICF + AdaBoost)	1.0×	1.3 fps
+ WaldBoost	2.6×	3.2 fps
+ Multiscale Approximations	17.5×	56.0 fps

Detector Performance. The detection quality of our detector on the TME Motorway Sequence is given as precision-recall curve in Figure 8. In Table 2 we report the detection speed of our detector. The speedup on per-component basis shows that both the soft-cascade and multiscale approximations are essential for reaching these speeds. Not surprisingly adding more scales is very cheap since by using multiscale approximations no image rescaling is required.

7. Discussion and Conclusion

In this work we have described the system design of our object detector. We show that it is possible to perform high-speed object detections at 56 fps on a laptop without including prior scene information or reducing the search space. Both WaldBoost and multiscale approximations are imperative for reaching these speeds. We acknowledge a decrease in detection rate of about 10% with our implementation of multiscale approximations. Possible explanations for this drop in performance are (1) the fact that our trainings images

are not available in high resolution, (2) some features are too small causing the feature approximations to be inaccurate and (3) detection thresholds for the strong classifier are not optimized and are currently the same for each scale.

For additional detection speedups we believe two strategies are the most promising: (1) streamline the detection pipeline and (2) for further work we could incorporate search space techniques to further speed up. For streamlining the detection pipeline we propose moving the classification task to the GPU. For devices without *shared memory* between CPU and GPU this has the advantage of not pushing images between CPU and GPU. Additionally this idea is motivated by the fact that subwindows can be classified independently hence massive parallelization offered by the GPU processing units is beneficial. An alternative idea for speeding up the detector would be to use the WaldBoost heatmap displayed in Figure 3 over a number of frames to build a *occurrence probability map* and include this as prior information for reducing the search space.

Beneson *et al*. [1] use stereo-vision to build a "stixel world" to reduce the search space dramatically. Their stixel world reduces the search space to 640×60 pixels $\cdot$ 10 scales. At this particular setting the authors report speeds of 135 fps on a high-end laptop featuring Intel Core i7 and dedicated GPU (faster than ours). As comparison we also reduced the search space by simply excluding top and bottom part of the image to 640×130 pixels $\cdot$ 25 scales. After this search space reduction our detector runs at 150 fps without sacrificing on detection quality. To our best knowledge our detector is one of the fastest object detectors reported in computer vision literature. For future work we are interested in a direct comparison in detection quality and explore different approaches for search space reduction.

References

[1] R. Benenson, M. Mathias, R. Timofte, and L. Van Gool. Pedestrian detection at 100 frames per second. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2903–2910. IEEE, 2012.

[2] R. Benenson, M. Mathias, T. Tuytelaars, and L. Van Gool. Seeking the strongest rigid detector. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 3666–3673. IEEE, 2013.

[3] R. Benenson, M. Omran, J. Hosang, and B. Schiele. Ten years of pedestrian detection, what have we learned? *arXiv preprint arXiv:1411.4304*, 2014.

[4] L. Bourdev and J. Brandt. Robust object detection via soft cascade. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 236–243. IEEE, 2005.

[5] C. Caraffi, T. Vojir, J. Trefny, J. Sochman, and J. Matas. A System for Real-time Detection and Tracking of Vehicles from a Single Car-mounted Camera. In *ITS Conference*, pages 975–982, Sep. 2012.

[6] 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*, volume 1, pages 886–893. IEEE, 2005.

[7] P. Dollár, R. Appel, S. Belongie, and P. Perona. Fast feature pyramids for object detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(8):1532–1545, 2014.

[8] P. Dollár, Z. Tu, P. Perona, and S. Belongie. Integral channel features. In *BMVC*, volume 2, page 5, 2009.

[9] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. 88(2):303–338, June 2010.

[10] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1627–1645, 2010.

[11] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[12] W. Gander. On halley's iteration method. *American Mathematical Monthly*, pages 131–134, 1985.

[13] 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.

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

[15] T. Lindeberg. Scale-space theory: A basic tool for analyzing structures at different scales. *Journal of applied statistics*, 21(1-2):225–270, 1994.

[16] M. Mathias, R. Timofte, R. Benenson, and L. Van Gool. Traffic sign recognition—how far are we from the solution? In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–8. IEEE, 2013.

[17] K.-Y. Park and S.-Y. Hwang. Robust range estimation with a monocular camera for vision-based forward collision warning system. *The Scientific World Journal*, 2014, 2014.

[18] V. Prisacariu and I. Reid. fastHOG-a real-time gpu implementation of HOG. *Department of Engineering Science*, 2310, 2009.

[19] D. L. Ruderman and W. Bialek. Statistics of natural images: Scaling in the woods. *Physical review letters*, 73(6):814, 1994.

[20] M. A. Sadeghi and D. Forsyth. 30hz object detection with dpm v5. In *Computer Vision–ECCV 2014*, pages 65–79. Springer, 2014.

[21] J. Shima. DSP Trick: Fixed-Point Atan2 With Self Normalization. 1994.

[22] J. Sochman and J. Matas. Waldboost-learning for time constrained sequential detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 150–156. IEEE, 2005.

[23] D. Tolhurst, Y. Tadmor, and T. Chao. Amplitude spectra of natural images. *Ophthalmic and Physiological Optics*, 12(2):229–232, 1992.

[24] A. Torralba and A. Oliva. Statistics of natural image categories. *Network: computation in neural systems*, 14(3):391–412, 2003.

[25] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[26] A. Wald. Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16(2):117–186, 1945.

[27] R. Xiao, L. Zhu, and H.-J. Zhang. Boosting chain learning for object detection. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 709–715. IEEE, 2003.

[28] C. Zhang and P. A. Viola. Multiple-instance pruning for learning efficient cascade detectors. In *Advances in Neural Information Processing Systems*, pages 1681–1688, 2008.

[29] Q. Zhu, M.-C. Yeh, K.-T. Cheng, and S. Avidan. Fast human detection using a cascade of histograms of oriented gradients. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1491–1498. IEEE, 2006.

Bibliography

[1] Roland Baddeley. Searching for filters with'interesting'output distributions: an uninteresting direction to explore? *Network: Computation in Neural Systems*, 7(2):409–421, 1996.

[2] Andre LC Barczak and Farhad Dadgostar. Real-time hand tracking using a set of cooperative classifiers based on haar-like features. 2005.

[3] Rodrigo Benenson, Markus Mathias, Radu Timofte, and Luc Van Gool. Pedestrian detection at 100 frames per second. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2903–2910. IEEE, 2012.

[4] Rodrigo Benenson, Markus Mathias, Tinne Tuytelaars, and Luc Van Gool. Seeking the strongest rigid detector. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 3666–3673. IEEE, 2013.

[5] Rodrigo Benenson, Mohamed Omran, Jan Hosang, and Bernt Schiele. Ten years of pedestrian detection, what have we learned? *arXiv preprint arXiv:1411.4304*, 2014.

[6] Christopher M Bishop et al. *Pattern recognition and machine learning*, volume 4. springer New York, 2006.

[7] Lubomir Bourdev and Jonathan Brandt. Robust object detection via soft cascade. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 236–243. IEEE, 2005.

[8] Claudio Caraffi, Tomas Vojir, Jura Trefny, Jan Sochman, and Jiri Matas. A System for Real-time Detection and Tracking of Vehicles from a Single Car-mounted Camera. In *ITS Conference*, pages 975–982, Sep. 2012.

[9] Franklin C Crow. Summed-area tables for texture mapping. *ACM SIGGRAPH computer graphics*, 18(3):207–212, 1984.

[10] Navneet Dalal. *Finding people in images and videos*. PhD thesis, Institut National Polytechnique de Grenoble-INPG, 2006.

[11] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[12] N G Deriugin. The power spectrum and the correlation function of the television signal. *Telecommunications*, 1(7):1–12, 1956.

[13] Piotr Dollár, Ron Appel, Serge Belongie, and Pietro Perona. Fast feature pyramids for object detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(8):1532–1545, 2014.

[14] Piotr Dollár, Ron Appel, and Wolf Kienzle. Crosstalk cascades for frame-rate pedestrian detection. In *Computer Vision–ECCV 2012*, pages 645–659. Springer, 2012.

[15] Piotr Dollár, Serge Belongie, and Pietro Perona. The fastest pedestrian detector in the west. In *BMVC*, volume 2, page 7. Citeseer, 2010.

[16] Piotr Dollár, Zhuowen Tu, Pietro Perona, and Serge Belongie. Integral channel features. In *BMVC*, volume 2, page 5, 2009.

[17] Piotr Dollár, Zhuowen Tu, Hai Tao, and Serge Belongie. Feature mining for image classification. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[18] Piotr Dollár, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: A benchmark. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 304–311. IEEE, 2009.

[19] Piotr Dollár, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: An evaluation of the state of the art. *PAMI*, 34, 2012.

[20] Andreas Eidehall. *An automotive lane guidance system*. Univ., 2004.

[21] Andreas Ess, Bastian Leibe, and Luc Van Gool. Depth and appearance for mobile scene analysis. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[22] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010.

[23] Pedro F Felzenszwalb, Ross B Girshick, and David McAllester. Cascade object detection with deformable part models. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 2241–2248. IEEE, 2010.

[24] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1627–1645, 2010.

[25] Pedro F Felzenszwalb and Daniel P Huttenlocher. Pictorial structures for object recognition. *International Journal of Computer Vision*, 61(1):55–79, 2005.

[26] David J Field. Relations between the statistics of natural images and the response properties of cortical cells. *JOSA A*, 4(12):2379–2394, 1987.

[27] David J Field. Wavelets, vision and the statistics of natural scenes. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 357(1760):2527–2542, 1999.

[28] Peter A Flach. The geometry of roc space: understanding machine learning metrics through roc isometrics. In *ICML*, pages 194–201, 2003.

[29] Luc Florack. *Image structure*. Kluwer Academic Publishers, 1997.

[30] Luc MJ Florack, Bart M ter Haar Romeny, Jan J Koenderink, and Max A Viergever. Scale and the differential structure of images. *Image and Vision Computing*, 10(6):376–388, 1992.

[31] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[32] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

[33] Walter Gander. On halley's iteration method. *American Mathematical Monthly*, pages 131–134, 1985.

[34] Jonas Gårding and Tony Lindeberg. Direct computation of shape cues using scale-adapted spatial derivative operators. *International Journal of Computer Vision*, 17(2):163–191, 1996.

[35] D Gavrilla. Daimler pedestrian benchmark data set. *follow "Looking at people" on http://www. gavrila. net/Research/research. html*, 2009.

[36] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[37] SG Ghurye. A characterization of the exponential function. *American Mathematical Monthly*, pages 255–257, 1957.

[38] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition*, 2014.

[39] Helmut Grabner, Jan Ŝochman, Horst Bischof, and Jiří Matas. Training sequential on-line boosting classifier for visual tracking. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4. IEEE, 2008.

[40] Marti A. Hearst, Susan T Dumais, Edgar Osman, John Platt, and Bernhard Scholkopf. Support vector machines. *Intelligent Systems and their Applications, IEEE*, 13(4):18–28, 1998.

[41] Roman Juránek. Detection of dogs in video using statistical classifiers. In *Computer Vision and Graphics*, pages 249–259. Springer, 2009.

[42] Zdenek Kálal. Diploma thesis face detection with waldboost algorithm. 2007.

[43] Michael Kearns. Thoughts on hypothesis boosting. *Unpublished manuscript*, 45:105, 1988.

[44] Jan J Koenderink. The structure of images. *Biological cybernetics*, 50(5):363–370, 1984.

[45] Jan J. Koenderink and Andrea J van Doorn. Generic neighborhood operators. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(6):597–605, 1992.

[46] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[47] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *Neural Networks, IEEE Transactions on*, 8(1):98–113, 1997.

[48] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361:310, 1995.

[49] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 05 2015.

[50] Tony Lindeberg. Scale-space for discrete signals. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 12(3):234–254, 1990.

[51] Tony Lindeberg. Scale-space theory: A basic tool for analyzing structures at different scales. *Journal of applied statistics*, 21(1-2):225–270, 1994.

[52] Philip M Long and Rocco A Servedio. Random classification noise defeats all convex potential boosters. *Machine Learning*, 78(3):287–304, 2010.

[53] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[54] Gábor Lugosi and Nicolas Vayatis. On the bayes-risk consistency of regularized boosting methods. *Annals of Statistics*, pages 30–55, 2004.

[55] Markus Mathias, Radu Timofte, Rodrigo Benenson, and Luc Van Gool. Traffic sign recognition—how far are we from the solution? In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–8. IEEE, 2013.

[56] Mayeul Mathias, Rodrigo Benenson, Radu Timofte, and Luc Van Gool. Handling occlusions with franken-classifiers. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1505–1512. IEEE, 2013.

[57] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2012.

[58] Eshed Ohn-Bar and Mohan Manubhai Trivedi. Learning to detect vehicles by clustering appearance patterns. 2015.

[59] Alan V Oppenheim, Alan S Willsky, and S Hamid Nawab. *Signals and Systems*. Pearson Education, 1998.

[60] Constantine P Papageorgiou, Michael Oren, and Tomaso Poggio. A general framework for object detection. In *Computer vision, 1998. sixth international conference on*, pages 555–562. IEEE, 1998.

[61] Ki-Yeong Park and Sun-Young Hwang. Robust range estimation with a monocular camera for vision-based forward collision warning system. *The Scientific World Journal*, 2014, 2014.

[62] Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, pages 1065–1076, 1962.

[63] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[64] Sobel Prewitt. Scharr gradient 5x5 convolution matrices guennadi (henry) levkine email: hlevkin at yahoo. com vancouver, canada. *First draft, February*, 2011.

[65] Victor Prisacariu and Ian Reid. fasthog-a real-time gpu implementation of hog. *Department of Engineering Science*, 2310, 2009.

[66] Sreeraman Rajan, Sichun Wang, Robert Inkol, and Alain Joyal. Efficient approximations for the arctangent function. *memory*, 4:5, 2006.

[67] Bart M Ter Haar Romeny. Introduction to scale-space theory: Multiscale geometric image analysis. In *Verlag. First International Conference on Scale-Space theory*. Citeseer, 1996.

[68] D Ruderman and William Bialek. Seeing beyond the nyquist limit. *Neural Computation*, 4(5):682–690, 1992.

[69] Daniel L Ruderman. The statistics of natural images. *Network: computation in neural systems*, 5(4):517–548, 1994.

[70] Daniel L Ruderman and William Bialek. Statistics of natural images: Scaling in the woods. *Physical review letters*, 73(6):814, 1994.

[71] Daniel Lee Ruderman. Natural ensembles and sensory signal processing. 1993.

[72] Mohammad Amin Sadeghi and David Forsyth. 30hz object detection with dpm v5. In *Computer Vision–ECCV 2014*, pages 65–79. Springer, 2014.

[73] Robert E Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of statistics*, pages 1651–1686, 1998.

[74] Robert E Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine learning*, 37(3):297–336, 1999.

[75] Bernhard Schölkopf and Alex Smola. Support Vector Machines. *Encyclopedia of Biostatistics*, 1998.

[76] Thomas Serre, Minjoon Kouh, Charles Cadieu, Ulf Knoblich, Gabriel Kreiman, and Tomaso Poggio. A theory of object recognition: computations and circuits in the feedforward path of the ventral stream in primate visual cortex. Technical report, DTIC Document, 2005.

[77] Jim Shima. DSP Trick: Fixed-Point Atan2 With Self Normalization. 1994.

[78] Bernard W Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.

[79] Jan Sochman and Jiri Matas. Waldboost-learning for time constrained sequential detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 150–156. IEEE, 2005.

[80] Elizabeth S Spelke. Principles of object perception. *Cognitive science*, 14(1):29–56, 1990.

[81] Anuj Srivastava, Ann B Lee, Eero P Simoncelli, and S-C Zhu. On advances in statistical modeling of natural images. *Journal of mathematical imaging and vision*, 18(1):17–33, 2003.

[82] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[83] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[84] S. Theodoridis and K Koutroumbas. *Pattern Recognition (4th edition)*. Academic Press, 2009.

[85] DJ Tolhurst, Y Tadmor, and Tang Chao. Amplitude spectra of natural images. *Ophthalmic and Physiological Optics*, 12(2):229–232, 1992.

[86] Antonio Torralba and Aude Oliva. Statistics of natural image categories. *Network: computation in neural systems*, 14(3):391–412, 2003.

[87] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[88] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[89] Paul Viola, Michael J Jones, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 734–741. IEEE, 2003.

[90] Abraham Wald. Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16(2):117–186, 1945.

[91] Rong Xiao, Long Zhu, and Hong-Jiang Zhang. Boosting chain learning for object detection. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 709–715. IEEE, 2003.

[92] Chun-Nam John Yu and Thorsten Joachims. Learning structural svms with latent variables. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1169–1176. ACM, 2009.

[93] Cha Zhang and Paul A Viola. Multiple-instance pruning for learning efficient cascade detectors. In *Advances in Neural Information Processing Systems*, pages 1681–1688, 2008.

[94] Qiang Zhu, M-C Yeh, Kwang-Ting Cheng, and Shai Avidan. Fast human detection using a cascade of histograms of oriented gradients. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1491–1498. IEEE, 2006.

[95] Daniel Zoran and Yair Weiss. Scale invariance and noise in natural images. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2209–2216. IEEE, 2009.