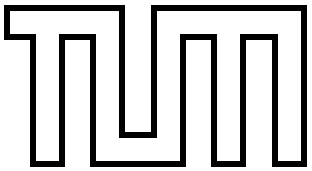


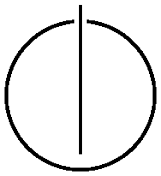
DEPARTMENT OF INFORMATICS

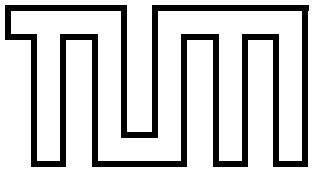
TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Pedestrian detection in urban environments based on vision and depth data

Andreas Kreutz





DEPARTMENT OF INFORMATICS

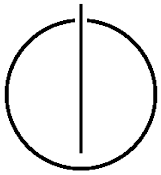
TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Pedestrian detection in urban environments based on vision and depth data

Personenerkennung in städtischen Umgebungen basierend auf Bild- und Tiefendaten

Author:	Andreas Kreutz
Supervisor:	Prof. Dr.-Ing. habil. Alois Knoll
Advisors:	Biao Hu, M.Sc.
Submission date:	March 15, 2017



I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

München, March 18, 2017 Andreas Kreutz

Abstract

Accurate pedestrian detection is important for many future technologies. Detection based on vision data is widely used in many applications today, however it seems like the maximum accuracy that can be achieved has almost been reached. Therefore, other ways of improving detection such as using depth data have to be considered.
In this thesis I will describe the Integral Channel Features detector, a detection algorithm based on visual data. I will also describe how the algorithm can be effectively trained using Adaptive Boosting techniques. Finally, I will give an overview of how depth data can be used to improve the performance of the detector.

Contents

List of Abbreviations

LiDAR	Light detection and ranging
ICF	Integral Channel Features
SVM	Support Vector Machine
HOG	Histogram of Oriented Gradients
fppi	false positives per image
fppw	false positives per window
LUV	(L*,u*,v*) color space
NMS	Non-Maximal Suppression
AdaBoost	Adaptive Boosting
RAM	Random access memory
CPU	Central processing unit
GPU	Graphics processing unit
TOF	Time-of-flight

1 Introduction

1.1 Motivation

Object detection is a very important topic of research in the field of image processing and machine learning. The goal of detection is to gather information about the real environment from sensor data. An accurate and fast detector forms the basis of any surveillance, guidance and collision avoidance system and will be essential for technologies such as self-driving cars, robotics and advanced human-machine interaction. In pedestrian detection, accuracy is even more important, since missed detections could in extreme cases lead to injury or death.

1.2 Problem statement

Most state-of-the-art algorithms today only use visual data of medium to low resolution for detection. This is because sensors that gather this type of data (mainly RGB cameras) are very cheap and have a small form factor, and therefore can be used in most use cases. However, using only a single type of sensor means that the performance of the detector is tied to the quality of the data gathered by this sensor. While today's cameras are very accurate in well-lit scenes with pedestrians that are clearly distinct from the background, their output deteriorates at night or in bad weather. To ensure that detection also performs well in these conditions, it is advisable to use more than one type of data in the detection algorithm.
Among other types of data, depth information should be considered. It is a good candidate to supplement visual data, because depth sensors are available in a variety of forms, ranging from cheap and relatively inaccurate stereo camera setups to much more accurate and thereby more expensive sensors such as light detection and ranging (LiDAR). The algorithms to evaluate depth data however are independent of the sensor. This means that it is possible for each use case to pick the sensor which is most appropriate. In cases where reliability is a concern, multiple sensor systems can also be used redundantly to improve their performance in difficult conditions.

1.3 Thesis structure

In this thesis I will describe an algorithm that can detect pedestrian based on visual data. I will then review a number of ideas how depth data could be used to improve the per-

formance of the detector.
In chapter 2 I will give an overview of related work. In chapter 3 I will start by explaining the basic ideas behind modern day object detection. In chapter 4 I will introduce the baseline detector that I used for my work and its special features. In chapter 5 I will present how the detector can be trained for accurate results and in chapter 6 I will show the results of training and detecting. In chapter 7 I will show how the baseline detector can be augmented with depth data to improve performance and in chapter 8 I will conclude.

2 Related Work

In this section I want to give an overview of the most important developments over the last 15 years that lead to modern day object detectors. I will also mention the most important datasets and benchmarks for pedestrian detection.
One of the first detectors that used a sliding window to scan images was developed by Papageorgiou and Poggio [17]. Their algorithm used Haar Wavelets to find similarities between different examples of humans and then trained a support vector machine (SVM) to recognise them in the sliding window.
Further contributions were made by Viola and Jones [22]. They introduced the integral image, an image representation that makes it very simple to calculate rectangular features. Their detector was trained using AdaBoost [12] to train a strong classifier that was able to find these features in images. Lastly, they cascaded the evaluation of the weak classifiers, so that less time was spend on windows that were not likely to contain humans.
Another important step was the proposition of gradient-based features by Dalal and Triggs[2]. These histograms of oriented gradients (HOG) were simple to compute and improved performance significantly.
Dollàr *et al*[6] examined the performance of different image channels. They concluded that using HOG, gradient magnitude and LUV color channels produced the best results. They also discovered that using an integral channel representation of the images combined with simple rectangular features did not impact performance and greatly increased the speed.
In later work, Dollàr *et al*[4] were able to make detection algorithms much faster. Their discovery was that one could compute a finely sampled image pyramid, which is necessary for detecting pedestrians of different sizes, by building a more coarsely sampled pyramid and then approximating the remaining scales.
Benenson *et al* [1] used this concept to achieve further speed improvements. They proposed a training method that creates a pyramid of classifiers, which can then be used to check an image for pedestrians of multiple scales without having to rescale the image.
All of the previously mentioned algorithm use a machine learning framework to detect pedestrians. To encourage progress in research, the datasets that are used for training and evaluation have to become more and more complex. Large datasets allow for better training and difficult datasets pose a challenge that researchers have to overcome. Therefore over the last decade the standard for datasets has changed.
Early person datasets such as the MIT dataset used by Papageorgiou and Poggio[17] contain cropped images that only show people without their surroundings. The goal of algorithms working with these datasets was classification rather than detection. This task is

comparatively easy and many of the datasets are effectively solved [2].
There are currently five large datasets for detection based on visual data, the INRIA [2], ETH Zürich [11], TUD Brussels [23] and Caltech [7] pedestrian datasets and the Daimler Detection Benchmark [8]. These datasets can be used for training as well as testing.
To be able to more easily compare results from multiple detection algorithms, a number of benchmarks were created that researchers can use to verify their results. The most popular benchmark is the Multiple Object Tracking Benchmark compiled by Milan *et al* [14] [16].

3 The Basics of Object Detection

Object detection as it was introduced by Papageorgiou and Poggio[17] relies on finding similarities in overcomplete representations of images. For this purpose, at first multiple image channels are computed on the input image. These channels are then evaluated by a learned model that can differentiate between pedestrians and background.

3.1 Creating an overcomplete image representation

Working directly with the pixel values of the input image is not sufficient to ensure accurate detection. For this reason, different detectors use a number of linear and non-linear transformations to calculate multiple representations of the input image. These representations or channels range from very simple transformations, such as grayscale and color channels, to more complex ones that capture edges or 'texturedness' of objects in the image [6].
While there are many candidates for channels, it is important to ensure that the transformations that created them are translationally invariant. This means that given a channel generating function Ω and given two images I and I' that are related by translation, the resulting channels $C = \Omega(I)$ and $C' = \Omega(I')$ must also be related by the same translation. This property allows a detector to efficiently find targets at different positions in an image, because Ω only has to be evaluated once for the entire image instead of once at each position [6].

3.2 Training a model for classification

To train an accurate model that is able to reliably classify, it is necessary to gather a sufficient amount of positive examples of pedestrians at a predetermined scale. In addition to that, a diverse collection of negative examples is needed. The training images should be similar to the data from the use case of the detector. For example, for pedestrian detection in urban environments, positive examples should show pedestrians in varied poses. Negative examples should be images of buildings, cars, trees and other pieces of scenery commonly found in cities.

For a successful training process it is necessary that the positive examples are at least somewhat similar, however still distinct enough to detect different manifestations of an object class. This means that while the viewing angles on objects in different examples

Figure 3.1: Evaluating two detection windows

should be varied, their scale and position has to be the same across all samples. The resulting model will be able to classify objects of a specific size and position in an image.
The classification model is trained for a predetermined size of images. This is called the detection window. An example for this is shown in figure 3.1. The model in the example was trained to classify windows of size 56×132. If there is a pedestrian in the center of such a window, the window is evaluated as true.
Any learning algorithm can be used to condition the model. The two most popular alternatives are Support Vector Machines (SVM) and Boosting algorithms [7]. Both are well understood and guarantee good performance.

3.3 Applying the classification model

Using the learned classifier, it is possible to detect objects in a single detection window. The size of this window depends on the dimensions of the positive and negative examples that were chosen during training. Most applications of detection however require the algorithm to find objects at all positions of the input image. In addition to that, objects that appear smaller because they are farther away from the sensor should also be detected. It is the task of the detection algorithm to apply the classification model in such a way that all objects in an image are detected.

3.3.1 Detection at different positions

Since in most applications, the detection window is much smaller than the input image, it is necessary to evaluate the classification model at multiple positions in the image. As in advance it is unknown where the targets in an image with dimensions $w \times h$ are, the detector has to scan every single column and row from $(0, 0)$ to $(w - window\_width, h -$

Figure 3.2: A sliding window detector

Figure 3.3: The pyramid of scales

$window\_height$). Figure 3.2 shows two steps of this process. The two steps in the figure are not consecutive, that means that a multitude of intermediate steps happen between the ones shown.
This so-called sliding-window approach ensures that no objects are missed [17].

3.3.2 Detection at different scales

During training, objects at only a single scale were examined so that patterns could be found accurately. This however means, that the sliding window will not be able to detect objects of a different scale. It is not possible to scale the results of the training without losing the learned information, because the patterns that can be recognized in images of different scales are quite different. Therefore the generally used approach for multi-scale detection is to instead scale the input image. A finely sampled pyramid of scales such as in figure 3.3 has to be constructed to guarantee that no objects are missed. The sliding-window detector then has to evaluate each scale sequentially [3].
An alternate approach to detection at different scales in conjunction with sliding-window detection was proposed by Benenson *et al* [1]. They showed that instead of rescaling the image to multiple different scales, it is possible to train a large number of classifiers, each able to detect objects of a different scale. This means that during runtime of the detector, much less time has to be spend on resampling the input image and computing the different channels. This way, they were able to achieve very high frame rates for their detector. Inversely however, the proposed approach has to spend a lot more time on training the pyramid of classifiers and also requires much more training data of pedestrians of different sizes.

3.3.3 Suppressing unlikely detections

The approach described so far is very capable of producing high detection rates. However since the sliding detection window moves with one pixel increments and the pyramid of scales has to be finely sampled, a single pedestrian in an image oftentimes result in multiple detections that are very close to each other or even have a great amount of overlap.

Figure 3.4: Four different results, from left two right: true positive, true negative, false positive, false negative

This detections are obviously superfluous, therefore most sliding-window detection algorithm employ some sort of suppression algorithm to remove them. [7]

3.4 Results of detection algorithms

After applying the classification model to all windows examined by the sliding-window on the pyramid of scales, and after suppressing unlikely detections, the result is a set of bounding boxes P of different sizes. Each bounding boxes has a confidence value > 0 that describes how likely it is that the detection is correct.
P can be separated into two subsets: True positives P_t are detections that contain an object from the object class, and false positives P_f are boxes that do not describe such an object. Implicitly, a detection algorithm also returns a set of negatives N, which can be divided into true negatives N_t (no object from the object class in this region) and false negatives N_f (missed detections). Examples for these different results are shown in figure 3.4 These sets are important for discussing the accuracy of different algorithms.

3.5 Evaluation of accuracy

Intuitively, an accurate detector returns large P_t and N_t, and small P_f and N_f. There are multiple metrics to quantize this thought and therefore to evaluate the performance of a detection algorithm.
Traditionally, precision and recall have been examined to measure the accuracy of pattern recognition. Precision is the proportion of true positives to all positives $\frac{|P_t|}{|P_t \cup P_f|}$, and recall

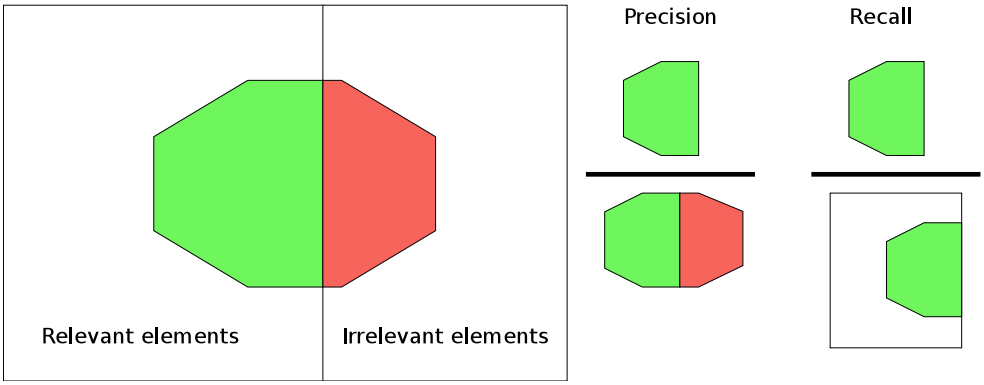


Figure 3.5: Precision and recall

is the ratio of true positives to all found and missed objects $\frac{|P_t|}{|P_t \cup N_f|}$. A more graphic definition of this metric is shown in figure 3.5.
In their evaluation of the state of the art, Dollàr *et al* [7] noted that there is an acceptable number of false positives per window in many tasks related to pedestrian detection. Therefore they decided to compare detection algorithms by plotting miss rate against false positives per image (fppi). For my work I will also use this metric, so that comparison of my results with the ones of other detectors is possible.

4 The Integral Channel Features Detector

For this work I decided to use the Integral Channel Features (ICF) detector introduced by Dollàr *et al* in [6] and improved in [5], [4] and [3]. This detector is a multi-scale sliding window detector. That means it is able to detect pedestrians of any size and at any position in an image. It uses AdaBoost with decision trees as the weak learners for training. The algorithm is the result of extensive testing and optimization and has state-of-the-art performance in terms of accuracy, while also having a very good frame rate [7].

4.1 Channel selection

To determine the best combination of channels, Dollàr *et al* [6] did a series of tests. The experiments were mostly performed on the INRIA pedestrian dataset introduced by Dalal and Triggs [2], which at the time was one of the largest and most challenging datasets for pedestrian detection.
To assess their findings, they used a similar metric to the one described previously, however they plotted false positives per window (fppw) instead of per image against the miss rate. Evaluating miss rate per window is better suited to compare the results of different classifiers instead of entire detection algorithms. However intuitively an improvement in per-window-performance of a classifier will also improve the per-image-performance of a detector using this classifier.

4.1.1 LUV color

Pixel value, or color is a likely candidate for a channel, since this data is the input of the detector and no transformation is required to calculate the channel. The choice of color model has some effect on the detection rate. The empirical research by Dollar *et al* [6] showed that with a detection rate of 55.8% at 10^{-4} fppw, the (L*,u*,v*) color space (LUV) model outperforms both the Red-Green-Blue (RGB) and Hue-Saturation-Value (HSV) models, as well as grayscale.

4.1.2 Gradient histogram

The gradient histogram channel attempts to capture the orientation of edges in an image. To achieve this, the image is first divided into cells and a gradient vector is calculated for each pixel in each cell, as shown in figure 4.1.

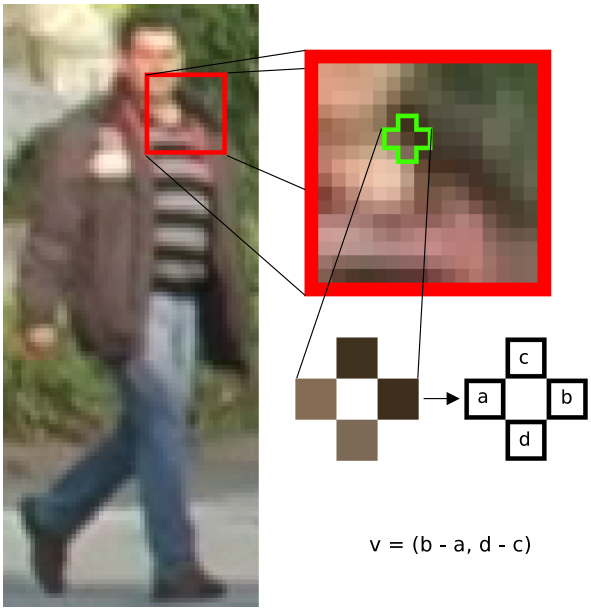


Figure 4.1: Calculating the gradient vector v for one pixel in a cell

Then, the vectors of each cell are binned into histograms by their orientation angle. Usually, the vectors are divided into 9 bins equally spaced in the range from 0° to 180°. This produced better results than spacing the bins from 0° to 360°, because pedestrians are very different from each other in terms of the color of their clothing. This means that the sign of the angle is not very informative, as it depends only on the value of two colors instead of the relation of their locations.
Finally, the histograms are normalized. Normalization is important because images or often very different in regards to illumination. This effects the brightness of each pixel and thereby the magnitude of the gradient vector. Normalizing removes this variance and makes it easier to find similarities. This step is performed on blocks of cells instead of on each cell separately. Normalizing is the key insight that makes gradient histograms the best channel currently for object detection, because a change of illumination of a scene does not change the gradients.
Although the idea of using features based on gradient orientation channels for detection was considered earlier, Dalal and Triggs [2] introduced significant improvements for pedestrian detection and formally described the algorithm explained above. Their Histogram of Oriented Gradients (HOG) detector achieves very high detection rates, yet for the ICF detector a simplified version of gradient histograms were used. Fundamentally both approaches work the same.
In their empirical research Dollàr *et al* found that the gradient histogram channel is the most accurate for detecting pedestrians among the ones tested. It achieved a detection rate of 87.2% with an average of 10^{-4} fppw. This is very close to the implementation

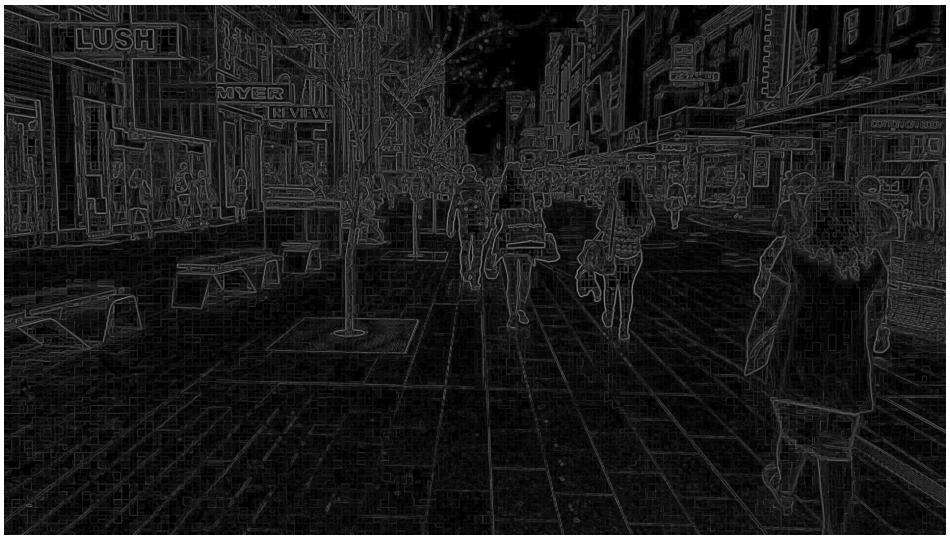


Figure 4.2: The gradient magnitude channel

of the HOG detector, which reached a detection rate of 89%. Through testing Dollàr *et all* also discovered that 6 bins are sufficient when computing the gradient histograms to achieve good detection. Using more than 6 orientations has little impact on performance.

4.1.3 Gradient magnitude

Because gradient magnitude has to be calculated anyways to build the gradient histogram channel, it makes sense to also use it as a channel for classification. Gradient magnitude is a representation of edges in an image, as seen in figure 4.2. Even though it does not carry much information on its own, combining this channel with the gradient histogram and LUV color channel improves the detection rate slightly.

4.1.4 Combining the channels and smoothing the data

By combining the three channels, Dollar *et al* were able to achieve detection rates of 91.9% with 10^{-4} fppw. In practice, the three channel types combine to a total of 10 channels, three for LUV color, one for gradient magnitude and six (one for each bin) for gradient histogram.
Smoothing the input data before calculating the channels slightly improves performance, as it can remove noise and small-scale structures in an image. This mostly affects the gradient channels, because smoothing or bluring the pixel values makes the gradient vectors in image regions more similar and continues. Dollar *et al* [6] determined that a Gaussian filter with a radius of 1 pixel works best and raises the detection rate by about 1.5% with 10^{-4} fppw.

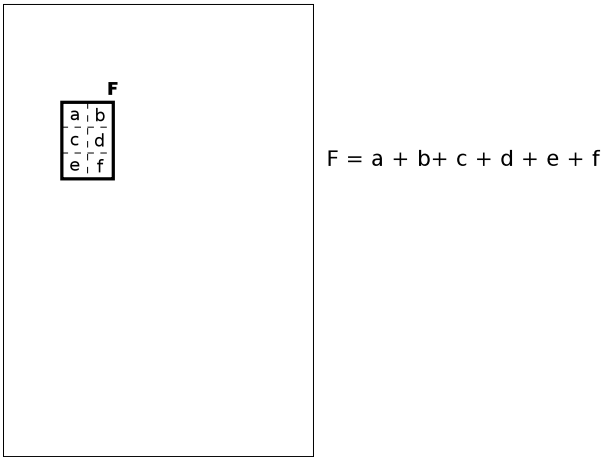


Figure 4.3: A first order features is a sum over a rectangular area of pixels

4.2 Evaluating channel data for detection

Instead of directly using the data from the previously described channels for detection, Dollàr *et al* [6] designed their detector to use a collection of features that aggregate pixel values of a channel over a region.

4.2.1 The concept of features

Using features was previously introduced by Viola and Jones [22] in their face detection algorithm. They used three kinds of features that could be computed by first calculating first-order features by summing the pixel values in rectangular areas on the channel data, as shown in figure 4.3. Next, higher-order features were computed as differences of two, three or four first-order features.
The experiments done by Dollar *et al* [6] resulted in a better understanding of the influence of using features on the detection rate. Most importantly, they discovered that using higher-order features does not provide a significant advantage over using first-order features.

4.2.2 Generating features

Since first-order features are less expensive to calculate and offer similar performance compared to higher-order features, the ICF detector uses only features of first order. The features are sums over rectangular areas with varying widths and heights. At first, Dollar *et al* [6] enforced a minimum area of 25 pixels for the rectangles, however in later work [5] they discovered that using large areas is not required for good detection rates. Instead it is more important to use intermediate size rectangles with a width and height between

8 and 16 pixels. Most rectangles smaller than 8 pixels per dimension also carry comparatively little information.
Features are evaluated on the detection window and on each individual channel. The number of available candidate features is

$$\sum_{8 \leq x \leq 16} \sum_{8 \leq y \leq 16} c \cdot (w - x) \cdot (h - y),$$

where c is the number of channels and w and h are the width and height of the detection window. This means that the number of possible candidate features a detector could use for classification is very high. For example, a detection window of size 56×132 pixels with 10 channels contains a total of 3 431 840 features. Evaluating all of them for each window during the detection step is not feasible, so during training some of these features have to be selected following some strategy. I will describe the training process in more detail in chapter 5.

4.2.3 Shrinking channel data

Figure 4.4: The shrinking process

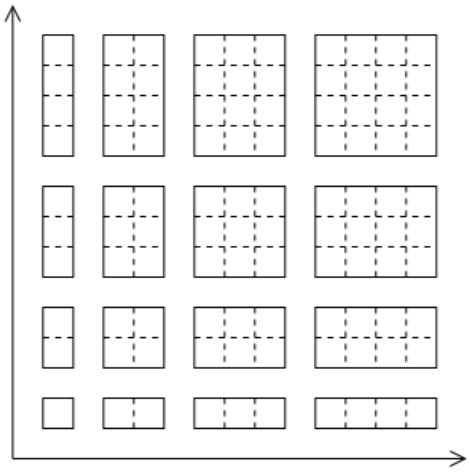


Figure 4.5: All feature sizes used

With their discovery that small feature are oftentimes not useful for classification, Dollar *et al* [5] concluded that it would be possible to shrink the channel data by a factor of four without decreasing performance. This means that the 56×132 pixels detection window is reduced to the size 14×33 and the width and height of feature rectangles now range from two to four. Evaluating a 4×3 feature on the shrinked detection window is equivalent to evaluating a 16×12 feature on the original one. An example is shown in

figure 4.4. By applying this method, features of intermediate size, such as 13×9 features, are also excluded from evaluation. However through testing, Dollàr *et al* found out that this does not result in a significant loss of performance.
The large decrease in the number of candidate features results in reduced training time. Because the time savings are very significant, for my work I decided to extend the range of sizes from $[2, 4]$ to $[1, 4]$ as seen in figure 4.5, so that the total number of available features now is $33\,480$. As a result, training will take longer, but the detection accuracy will be slightly better.

4.2.4 The integral image representation

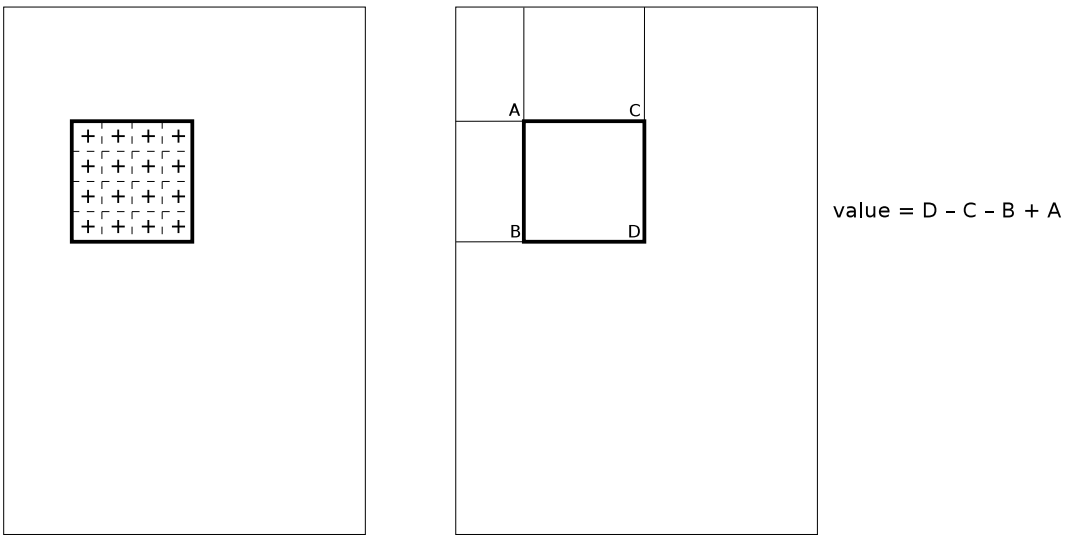


Figure 4.6: Two ways of calculating the value of a 4×4 feature: 16 additions on the left and 2 additions, 2 subtractions on the right

Evaluating features is a step in the process of detection that happens a large number of times. It therefore makes sense to optimize the calculation of feature values.
Features are sums over rectangular areas of pixels. If these sums were calculated separately for each feature, classification would take far too much time for large features, such as 4×4 features, which would require 16 lookups in the channel data.
To avoid this high amount of computation, Viola and Jones [22] proposed a different representation of channel data called the integral image. Each pixel (x, y) in this representation is the sum of all pixels above and to the left of it:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y'),$$

Figure 4.7: The bottom features is less informative than the top feature

where $i(x, y)$ is the value of the image channel at position x, y. By using the following equations, the integral image can be computed in $\mathcal{O}(width * height)$:

$$s(x,y) = \begin{cases} s(x,y-1) + i(x,y), & \text{if } y \geq 0. \\ 0, & \text{otherwise.} \end{cases}$$

$$ii(x,y) = \begin{cases} ii(x-1,y) + s(x,y), & \text{if } x \geq 0. \\ 0, & \text{otherwise.} \end{cases}$$

After constructing the integral image, all feature values for this image scale can be calculated with only four array lookups. Figure 4.6 shows an example for evaluating a 4×4 feature. Using this image representation drastically reduces the time it takes to find the value of a features and improves the runtime of training and detection.

4.2.5 Advantages of using features over pixel values

Using features instead of pixel values for detection is motivated primarily by considerations about how information about the content of a window can be retrieved. A lot of the data that is relevant for classification is very dependent on local contexts. Therefore it makes sense to try to find patterns in regions of pixels, and features are a good way of doing that.
Another insight into information retrieval is that different regions in a detection window contain different amounts of useful information, as shown in figure 4.7. Generating all regions and then picking the best from them is therefore a good way to build a classifier that is accurate as well as simple. In combination with features, the training algorithm (adaptive boosting) is able to create a classification model that fulfills these properties.

The resulting model will be made up of the most informative features available.

4.3 Building a pyramid of scales

To detect pedestrians of all sizes, the input image has to be scaled multiple times. The detector then searches each scale separately. In this chapter I will describe how constructing a finely sampled pyramid of scales can be done and how the traditional approach can be improved.

4.3.1 Number of scales required

Determining the number of scales necessary is essential. If it is too small, pedestrians can be missed but the more scales that are considered by an algorithm, the longer channel computation and detection will take. An important criteria in this discussion is the octave, which is the interval between scale 2^x and scale 2^{x+1} (or 2^{x-1} respectively). Three parameters should be examined, the number of scales per octave, as well as the number of octaves above and below the scale of the input image:

- Number of scales per octave
  This number is the deciding factor in making sure that no detections are missed. Dollar *et al* [4] used 8 scales per octave, which corresponds to a scale step size of approximately 1.09. This means that each subsequent scale is $1.09\times$ larger as the one before it.

- Number of octaves for upscaling
  This determines the upper bound for the size of pedestrians that can be detected. How many are necessary depends mainly on the use case of the detector. For pedestrian detection in urban environments, it makes sense to only upscale the height of the input image to $16\times$ the height of the detection window. Pedestrians that are smaller than that are too far away to be of concern.

- Number of octaves for downscaling
  This defines the lower bound for detection. The smallest possible scale is defined by the size of the detection window. An image scaled below this size cannot be classified by the sliding window detector.

Another thing to keep in mind is the resolution of the sensor used to acquire the input data. When working with images of low resolution, to much upscaling makes the data unusable as it becomes heavily influenced by noise and artifacts of the resizing algorithm.

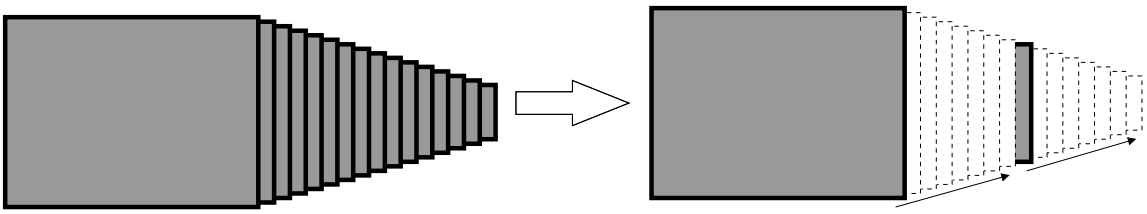


Figure 4.8: Left: No approximation, 16 scales have to be calculated
Right: Only 2 scales have to be calculated, the rest is approximated

4.3.2 Approximating scales in an octave

Building the finely sampled pyramid of scales is the bottleneck for many detection algorithms. For the ICF detector with a window width of 56×132, an input image of size 1920×1080 would have to be upscaled by one and downscaled by four octaves according to the previously described specifications. This means the image has to be resized 40 times and all 10 channels have to be computed on every single scale.
Computing the channels is the more time consuming part of the algorithm, so a solution to this problem is that instead of rescaling the image and then calculating channels, channels themselves are rescaled. Dollàr *et al* [3] investigated how a change of scale effects the channel data chosen for the ICF detector. They concluded that all scales in an octave could in fact be approximated from a single one that is computed explicitly without losing much accuracy. This insight allows detection algorithms to run an order of magnitude faster, with only an insignificant loss in accuracy.
A channel value $C = \Omega(I)$ for an image I is the result of a shift-invariant transformation Ω. The scaled image I_s with the smaller scale s can be calculated as $I_s = R(I, s)$, where R changes the dimensions $w \times h$ of I to $(w \cdot s) \times (h \cdot s)$. Traditionally, C_s at a different scale is thereby computed as $C_s = \Omega(R(I, s))$. It seems intuitive that C_s could also obtained by simply downscaling C. However, this is not true for channels based on gradient vectors, because reducing scale results in a loss of high frequency information. Losing high frequencies means a decrease of vector magnitude beyond the expected value s. However, this divergence is consistent in regards to scale and Dollàr *et al* discovered that the values of C and C_s are correlated by a power law:

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

The constant λ_Ω depends on the type of channel used, but is independent from the current scale that is calculated, and can be determined empirically [3]. This equation only applies for downscaling, i.e. $0 < s < 1$, as upscaling preserves all existing image information.
Using this insight allows finely sampled image pyramids to be constructed by explicitly calculating much less scales, as shown in figure 4.8. In fact, only one scale per octave is necessary to approximate the remaining ones. Because the power law only applies for

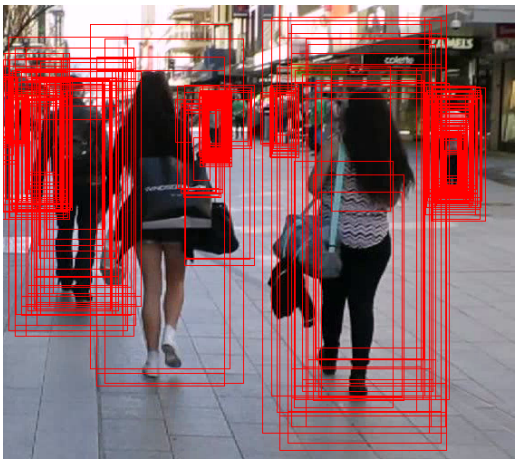


Figure 4.9: Detection result without NMS, most boxes can be removed

downscaling, the approximation within an octave is always computed in regards to the next real scale above the octave.

4.4 Evaluating features on the sliding window

Further optimization can be achieved by reducing the number of windows that are evaluated by the classifier. A traditional sliding window detector scans every single column and row of every single scale, thereby evaluating a very large number of windows. In practice however it is not necessary to check all of these windows, because the contents of consecutive windows are almost the same except for one row or column of pixels. Dollàr *et al* [6] therefore used a step size of 4 pixels for the ICF detector.

4.5 Non-maximal suppression

After the channel data is evaluated by the feature classifier, a number of detections in form of bounding boxes, each with a confidence value, is returned. Even with a step size of 4 pixels, a lot of these boxes have a significant overlap, which means that it is likely that they describe the same detected pedestrian. To remove these obviously superfluous detections, a simplified form of non-maximal suppression (NMS) is used. NMS compares each pair of bounding boxes and, if their area of overlap is larger than 60%, removes the one with the lower confidence value [6]. As a result, some errors that are intrinsic to the method of sliding window detection are already removed in the detection step. Figure 4.9 shows the output of the ICF detector without NMS. Many boxes clearly describe the same detection, because the overlap is very large, therefore they can be removed.

5 Feature Learning with Adaptive Boosting

The design of the ICF detector lends itself to the use of Adaptive Boosting (AdaBoost) for learning. AdaBoost is able to aggregate a large number of weak learning algorithms into a strong learner, that is able to classify detection windows accurately into positive and negative ones, depending on if they contain a pedestrian. The training process involves multiple rounds of bootstrapping to improve the accuracy further.

5.1 Definitions

To make describing the learning process easier, this section will establish a couple of definitions and naming conventions.

- Training the classification model is a supervised learning process. To achieve this, a collection of training images X and set of labels $Y \in \{1, 0\}$ is provided. Training examples are labeled instances in the form of $X \times Y$. The label $y_i \in Y$ determines whether $x_i \in X$ contains a pedestrian or not.
- A classifier is a function $h : (x, y) \rightarrow [0, 1]$. If $h(x, y) = y$ applies, the classification was correct.

5.2 Obtaining training examples

A number of researchers in the past have recorded footage of pedestrians in different environments and annotated their locations in the images to gather training examples for supervised learning. I have used the INRIA pedestrian dataset [2] and the TUD-Brussels dataset [23] for the training procedure.

5.2.1 Datasets

INRIA is one of the earliest large detection datasets publicly available. It contains upright persons in various environments, not restricted to urban areas. The training set consists of 614 positive images containing 1086 pedestrians in slightly varied poses, as well as 1218 negative images, mostly of cities and buildings but some of scenes of nature.
TUD-Brussels was created later and focuses on pedestrians in urban environments. The persons shown in the images are very typical in their behaviour to pedestrians in a city.

This data set is therefore very well suited for the purpose of pedestrian detection. It contains 1092 positive images of 1092 pedestrians, in addition to 384 negative images. The content of the negative images is not as varied as the images from INRIA.

5.2.2 Generating positive and negative training examples

The images from the two training datasets are used as positive and negative examples. To increase the number of positive windows, each window is flipped along the y-axis and both versions are added to the training set. $(1086 + 1092) \cdot 2$ positive images are therefore available for training.
Negative windows are generated by creating cutouts the size of the detection window at random positions in the negative images. Each image is used to generate 9 negative examples. In addition to that, a number of windows are obtained at 9 evenly spaced positions in the images. This way, all relevant parts of the images are likely to be included in the training set, while keeping the number of negative windows relatively small. In total, $28\,836$ negative windows are available.
However using all of them for training would still make the process take far too much time, so initially, two times the number of positive windows is used for training. The remaining negative windows are necessary for bootstrapping.

5.3 The concept of boosting algorithms

Boosting is a way of training a model to distinguish patterns in data. The idea is to combine the results of a number of weak classifiers into a strong classifier. Most often this is accomplished by giving each learner a weighted vote and considering all votes for the final classification. Since different weak learners rely on different parts of the information in the input data, this will lead to improved results.
Selecting classifiers that lower the overall error rate is encouraged by changing the set of learning examples trained with. In each iteration, $X \times Y$ is assigned a probability distribution that influences which examples are used in this step. Then, this distribution is altered so that examples that were falsely classified are more likely to appear in later steps. After each step, the classification problem becomes harder, because easy examples have a lower probability of appearing [12].

5.4 Generating weak learners

Since the number of features is very high, it makes sense to use them as the weak learners. Each feature can be turned into a decision tree stump by assigning a threshold to it. If the feature then evaluates to a value lower than the threshold, the window is classified as negative, or positive otherwise. Finding these thresholds and using them to create weak

Listing 5.1: Finding feature thresholds

```
bestErrorRate = 1;
bestThreshold = 0;
for (Image learningExample : learningExamples):
   featureValue = evaluateFeature(feature, learningExample);
   for (Image learningExample : learningExamples):
      errorRate = testThreshold(featureValue, learningExamples);
      if (errorRate < bestErrorRate):
         bestErrorRate = errorRate;
         bestThreshold = threshold;
```

classifiers is done before running AdaBoost, so that the algorithm then can pick from the set of weak learners during each step.

5.4.1 Finding the feature thresholds

The performance of the strong learner can be improved by finding good feature thresholds, thereby making the weak learners more accurate. Using supervised learning with a large set of positive and negative examples ensures that meaningful thresholds are found. The process of finding the threshold for one feature is described in listing 5.1. The runtime of this brute-force algorithm can be improved by sorting the thresholds before evaluating their error rates. It also makes sense to quantize the features once and save the results, so that $evaluateFeature$ only has to be executed once per feature and learning example.
The AdaBoost algorithm requires the weak learners to each have a better error rate than 0.5, that means weak learners have to perform at least slightly better than random guessing. Therefore, features that do not fulfill this property are discarded.

5.4.2 Choosing a subset of features

Training would take far too long if all available features were considered. Therefore, only a subset of them are chosen to aggregate into a strong learner. Finding the features that allow the classifier to perform optimally is not trivial. Figure 5.1 illustrates the problem. Features with low error rates oftentimes rely on similar information about the image, so it can be expected that boosting on them will not improve the overall accuracy by much. However picking features that are evenly spaced out over all rectangle sizes, positions and channels of the detection window does not guarantee that each feature has a good detection rate. Careful feature selection by hand could combine these two considerations, however because of the large number of candidates this is not feasible. A good solution is choosing the subset randomly, as this ensures that the set of features is not biased towards either of these categories.

Figure 5.1: The red and the green feature both contain very similar information, the yellow and the blue feature are spaced out, but not both are informative

5.4.3 Building classifier trees

Each feature of the set together with its threshold can be seen as depth-0 decision tree. These decision stumps can be directly used as the weak learners for AdaBoost, however a performance gain can be achieved by combining multiple of them into a deeper decision tree. With increasing depth the classification error of each weak learner decreases, thereby improving the total accuracy of the strong learner. However deeper trees will also lead to an increase in evaluation time. A good balance between accuracy and runtime is reached by using depth-1 decision trees.
Finding the best features to place at each node of the tree is another task were supervised learning can be used. Since in the last step the number of candidate features in the set of features F was greatly reduced, a fairly simple algorithm can be used:

1. Pick a random feature f_r as the root node, remove it from F
2. Evaluate all training examples with this feature, find the examples that are classified as false positives $E_{fp} := \{(x, 0) \mid f_r(x, 0) = 1\}$ and false negatives $E_{fn} := \{(x, 1) \mid f_r(x, 1) = 0\}$
3. For each feature f_i in F:
   a) Evaluate f_i for all training examples in E_{fp} and count the number of images n_{fp} that are also evaluated as false positives by this feature
   b) Repeat for E_{fn} to find all n_{fn}, the number of images that evaluate as false negatives
4. Find f_{fp} with lowest n_{fp} and f_{fn} with lowest n_{fn}

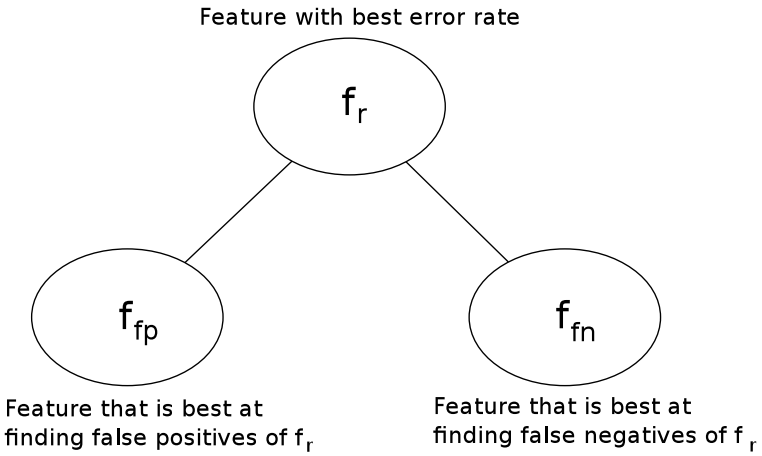


Figure 5.2: A weak classifier

5. Add f_{fp} as the left child of f_r and f_{fn} as the right one, remove them from F

The resulting tree as shown in figure 5.2 has an improved accuracy on the training set than the root node on its own. This is because the features at the child nodes are picked by their error rate on the subset of training examples that were missclassified by the root node.
This algorithm is repeated until a sufficient number of decision trees, for example 1000, is built. It is similar in concept to the C4.5 algorithm developed by Quinlan [20]. However C4.5 is much more efficient and uses additional techniques such as pruning the tree after its creation. This is not necessary for the purpose of building weak learners, because only decision trees with depth 1 have to be constructed.

5.5 Training the strong learner

The Adaptive Boosting algorithm used for training a strong classifier was formulated by Freund and Schapire [12]. Its advantage over other forms of boosting is that it does not rely on prior information about the performance of the weak learners. Instead, it dynamically calculates and assigns the weights of the classifiers depending on their performance on the set of learning examples. Freund and Schapire were able to prove that its training error is bounded and decreases with the number of training rounds.
For boosting, each example $(x_i, y_i) \in X \times Y$ is assigned a weight $w_{t,i}$ for each training step t. Weights are initialized to

$$w_{0,i} = \begin{cases} \frac{1}{2p}, & \text{if } y_i = 1. \\ \frac{1}{2n}, & \text{otherwise.} \end{cases}$$

where p, n are the number of positive and negative examples respectively. Each iteration t of AdaBoost then works like this [12]:

1. Normalize the weights so that they represent a probability distribution (they sum to one):
$$w_{t,i} = \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$

2. Evaluate all weak classifier h_j from the set of previously trained weak learners and save their results e_i for each learning example (x_i, y_i):
$$e_i = |h_j(x_i) - y_i| = \begin{cases} 0, & \text{if } h_j \text{ was correct,} \\ 1, & \text{otherwise} \end{cases}$$
Then calculate their errors $\epsilon_{t,j}$ according to $w_{t,i}$:
$$\epsilon_{t,j} = \sum_i w_{t,i} \cdot e_i$$

3. Choose $h_t = h_j$, $e_t = e_j$ and $\epsilon_t = \epsilon_j$ as the classifier with the lowest error ϵ_j

4. Update the weights:
$$w_{t+1,i} = w_{t,i} \cdot \beta_t^{1-e_t},$$
where $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$. Using β_t as a function of ϵ_t means that examples that were correctly classified have their weights reduced.

5. Give h_t an α-value $\alpha_t = \log \frac{1}{\beta_t}$

The number of iterations T determines how many weak learners are combined. The final strong classifier is
$$h(x) = \begin{cases} 1, & \sum_{t=0}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=0}^{T} \alpha_t \\ 0, & \text{otherwise.} \end{cases}$$

This means that each weak classifier h_t casts a vote in $[0, 1]$. Each vote is then weighted by the classifier's accuracy α_t. If the sum of weighted votes is greater or equal half of the total weight, x is evaluated to 1, or 0 otherwise.
In early rounds, h_t will have small error ϵ_t and therefore a large α_t. In later rounds, when the task of classifying $X \times Y$ becomes harder, ϵ_t will approach 0.5 and α_t will approach 0. At this point, the algorithm can terminate, because the strong classifier will not be improved by adding more voters.
The idea of using AdaBoost to choose the optimal weak classifiers from a set of features was previously introduced by Viola and Jones [22].

5.6 Bootstrapping

A technique to greatly improve the detection rate of the trained classification model is bootstrapping. It refers to the method of using the results of one detector to train a second one. In practice, this means that training using a set of positive windows P and negative windows N happens in several rounds. In each round t, a strong classifier h_t is trained using all windows in P, but only a subset $N_{current}$ of images in N. Then, h_t is used to evaluate all windows in N. Windows n_i that are missclassified, i.e. $h_t(n_i) = 1$, are added to $N_{current}$ and a new strong classifier is trained.
Because after each round more difficult windows are added to the training set, classifiers trained in subsequent rounds will perform better than the previously trained ones. The number of difficult windows added each round and the number of rounds depends on the training data used and on the time that is available for training. Adding more windows and training in more rounds greatly increases the runtime of training. The number of rounds is also limited by how many negative windows are available in N. For my work I used three rounds of bootstrapping.

5.7 Outcome of training

The training process conditions a classification model that is able to determine whether a window contains a pedestrian or not. This model can then be used in the framework of a sliding-window detection algorithm to find pedestrians in an image.
The training program creates two files to describe the strong classifier. The *features* file describes the individual rectangles and their thresholds. The features are stored so that they can easily be calculated on the integral image representation. Therefore each feature consists of the top-leftmost and bottom-rightmost point of the rectangle, the channel identifier on which the feature has to be evaluated, and a threshold value.
The *classifier* file contains the weak classifiers that AdaBoost selected. Each weak classifier is a depth-1 decision tree with an α value. This file references the *features* file, as the criterion tested for at each node of the tree is the evaluation of a feature. About 50 weak classifiers are enough for accurate results.

6 Results

In this chapter I will talk about my results of training a classification model and using it for detection. I employed a C++ implementation of the ICF detector proposed by Dollàr *et al* in [6]. I also closely followed their description of how to train the detector, which I summarized in the previous chapters.

6.1 Description of the hardware used

All tests were done entirely on an Intel i5 4200m processor, with a maximum frequency of 3.10 GHz, two cores, and 8 gigabytes of random access memory (RAM).
To speed up the training process, I employed multithreading in all parts of the procedure, however with the described hardware the algorithm is limited to only being able to run four threads in parallel. Using a central processing unit (CPU) with more cores or even employing a graphics processing unit (GPU) would speed up training tremendously.
Multithreading would also be of use in improving the runtime of the detection algorithm.

6.2 Training

In this section I will describe the parameters I used for training. I will also talk about the runtime of the procedure, as this is the most important consideration during training. The performance of the conditioned classification model in the detection task will be described in section 6.3.

6.2.1 Parameters

Training was done using two datasets, the INRIA pedestrian and the TUD Brussels set, for a total of $5\,968$ positive and $28\,836$ negative windows. I performed three rounds of bootstrapping, starting with $5\,968$ positive and negative windows in the first round and adding $5\,968$ difficult negative windows in each subsequent round.
In each round I not only retrained the strong classifier with AdaBoost, but also constructed the weak classifiers anew. This includes finding optimal thresholds for all features in each round.
Prior to boosting, $5\,000$ features were picked randomly from all usable ones. These features are then used to build $1\,000$ weak classifiers.

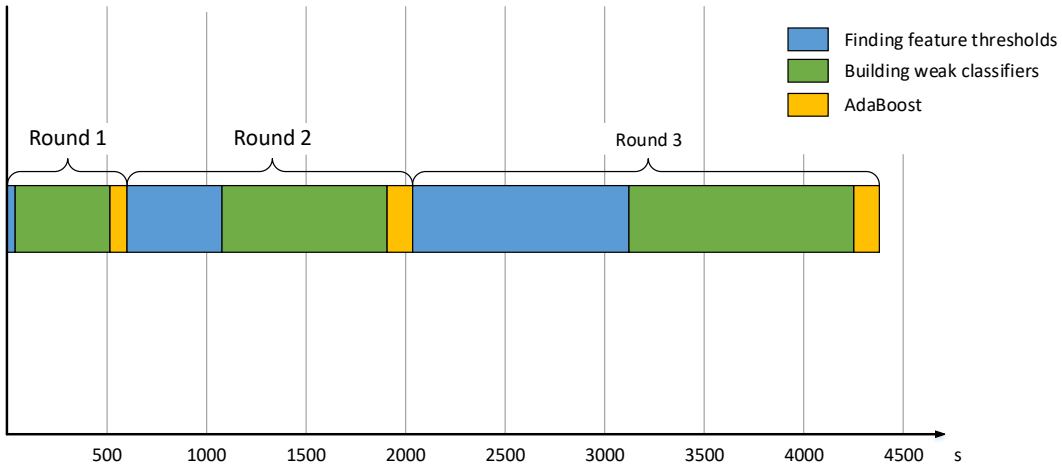


Figure 6.1: Composition of training time in seconds

6.2.2 Results

Training a classifier from start to finish takes about 4383 seconds on the previously described hardware. Figure 6.1 shows a summary of how the time was spend. Each round takes exponentially more time then the previous ones as more windows have to be considered.
The most time-consuming part of training is evaluating all features on the training images, because it happens a large number of times in all three steps of training, finding feature thresholds, building weak classifiers and boosting. This step can be sped up by quantizing all features once on all images and then saving the results. However for the extent of training I performed this was not possible, because the memory required to save the response of ~30 000 features on ~35 000 images is much larger then the 8 GB of RAM I had available.

6.3 Detection

In this section I will talk about the performance of the trained model in the context of the ICF detection algorithm in terms of accuracy as well as speed.

6.3.1 Accuracy

My focus for the evaluation of accuracy is per-image performance, that means how good a detector is at finding pedestrians in full size images. This criterion tests all parts of the

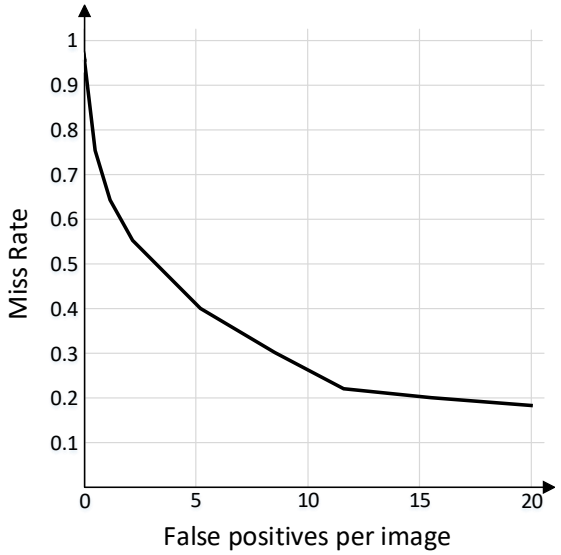


Figure 6.2: Miss rate of the trained ICF detector, lower is better

detection algorithms at once. It is an important metric to determine the performance of an algorithm in a real world setting.
Figure 6.2 shows the accuracy of the ICF detector with the classification model learned using my implementation of training. Tests were done with images from the test sets of the INRIA and TUD-Brussels datasets. At 10 false negatives per image, the model achieves a miss rate of 25%, which means that 75% of pedestrians are correctly detected.
A typical result of running the detector with the previously trained classification model on a 1920×1080 image can be seen in figure 6.3. All pedestrians at medium range are detected, however the algorithm has difficulties finding people that are very close to the camera and those that are far away. The degree of occlusion is also a deciding factor if a person is found. Finally, because the detector was trained for upright pedestrians, it is inaccurate for people that have a different pose, such as sitting. The example is from the Multiple Object Tracking Benchmark, which is much more difficult than the INRIA and TUD-Brussels sets.

6.3.2 Speed

The C++ implementation of the ICF detector runs on the previously described hardware at about 0.5 frames per second. This means that the detection on each input image takes 2 seconds to complete. For some use cases, this execution time will be enough. However in situations where objects in a scene change their positions by a lot in each frame, as for example in high speed automotive scenarios, a better runtime is required.

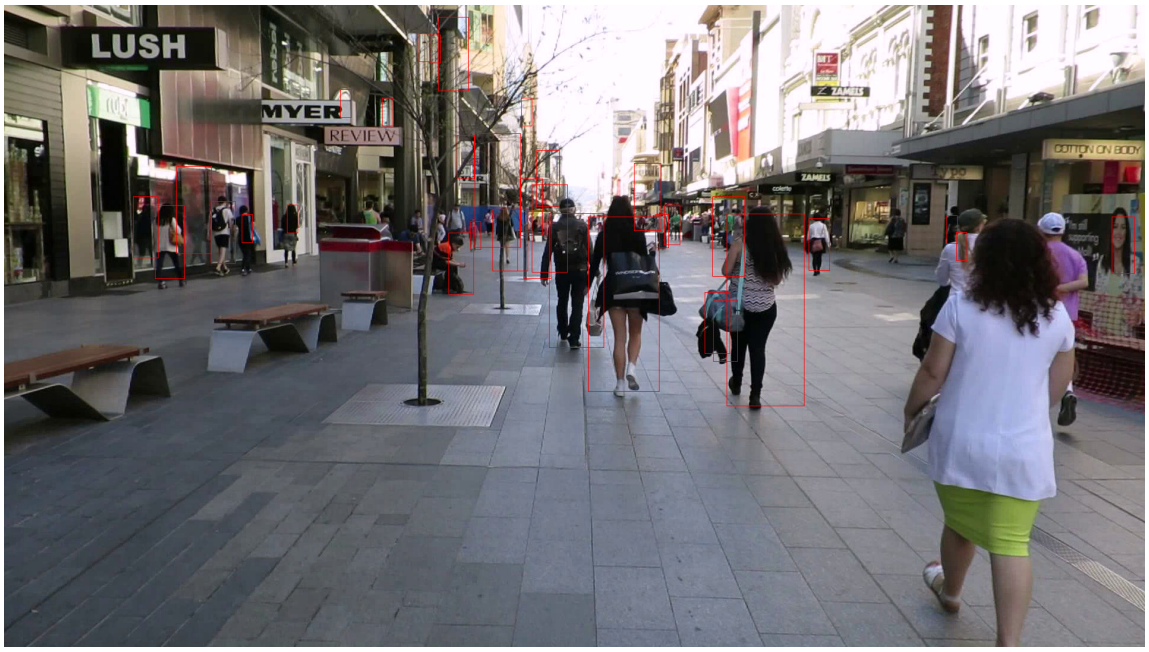


Figure 6.3: Detection result on a 1920×1080 image

Speed was not the main concern of this thesis, therefore there are still many ways how it can be improved. The algorithm is somewhat optimized, however it runs entirely single-threaded, which could easily be changed as the program is very suitable for parallelization. I will discuss this further in chapter 8.

7 Detection Based on Depth Data

Pedestrian detection based on visual data has been a topic of research for a long time. Many promising discoveries have been made that were able to improve detection accuracy as well as speed, so that state-of-the-art algorithms of today achieve very good results.
Because it is difficult to further improve performance based just on vision data, using other types of data such as depth information for detection has become an interesting point of research.

7.1 Obtaining depth information

Depth data in combination with visual data is often gathered in the form of depth maps, which map distance from the camera to each pixel of an image. There are multiple ways of obtaining depth maps, many of which have been used for pedestrian detection. This section will give an overview over different methods used as well as their advantages and disadvantages.

7.1.1 Stereo cameras

One of the cheapest ways of gathering depth data is by setting up two visual sensors in a stereo configuration. The goal is to obtain two images of the same scene that are taken from the same plane and from the same vertical level. Then, the two images are searched for matching pixels. The horizontal difference between the pairs of pixels can then be used to calculate depth maps (see figure 7.1).
This method of obtaining depth data is very simple and works well in addition to visual based detection, because the only additional sensor needed is a second camera. However completing the task of stereo matching is computationally expensive and oftentimes not very accurate. Matching algorithms work best on images with high contrast and clearly visible structures, however scenes of pedestrians in urban environments most often do not fulfill these properties. Additionally, because stereo matching relies on the same sensors as detection algorithms based on visual data, it will not improve performance in lighting conditions that are problematic for these detectors.
Stereo matching was for example used by Ess *et al* in their work on pedestrian detection and tracking [9] [10].

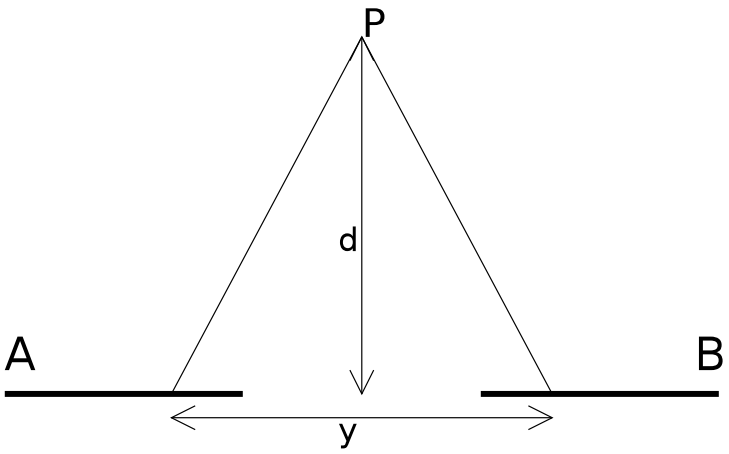


Figure 7.1: Depth d of a point P can be calculated from the horizontal difference y on two image planes A and B

7.1.2 Time-Of-Flight cameras

Time-of-flight (TOF) cameras are able to gather RGB-D images by measuring the time it takes for a light impulse send out by the device to return to each pixel of the sensor. Because they can capture the entire image at once, TOF cameras are able to work at very high frame rates.
As visual and depth information is captured by the same sensor, TOF cameras would be very suitable for pedestrian detection. However, most of these cameras are limited by having a low image resolution and only being accurate at short ranges, because they need to be able to detect the light impulse at every pixel. This also makes them not very useful outdoors, since the sun is a much stronger light source that drowns out the impulse emitted by the camera.
A TOF camera that is widely available today is the Kinect by Microsoft. It was used for indoor object detection, and Luber, Spinello and Arras used it in their work on people detection and tracking [15] [21].

7.1.3 Light Detection and Ranging

Light detection and ranging (LiDAR) sensors work similar to TOF cameras in that they measure the time it takes for a emitted beam of light to return. However this sensor does not capture the entire image at once and instead moves the beam around the scene to capture depth information.
LiDAR provides data of high resolution and gives precise data in a wide variety of ranges. The depth data is sparse however, that means not all pixels can be mapped to a depth value. Therefore some kind of interpolation is necessary for good results. LiDAR also is

not dependent on good scene illumination because it actively emits light, but problems can arise when materials with unusual reflective properties are encountered.
LiDAR sensors were among others used by Premebida *et al* [18] in their work on pedestrian detection for automotive purposes.

7.2 Datasets containing depth data

The limiting factor for research on depth data for pedestrian detection for a long time has been the lack of a publicly available dataset containing depth information with a large number of pedestrian bounding boxes to enable thorough training [18]. Researchers therefore have to collect their own data to verify their work. This makes it difficult to compare results and limits the usefulness of the proposed detection algorithms.
In 2013, Geiger *et al* [13] changed this and collected footage of urban environments using a set of stereo cameras as well as a LiDAR sensor. With this dataset publicly available, it can be expected that new and improved detectors based on visual and depth data will appear in the coming years.

7.3 Using depth data to improve detection algorithms

In this section I will present ideas on how depth data can be used to improve pedestrian detection. I will focus on approaches that can be combined with the previously described ICF detector.

7.3.1 Histogram of Oriented Depths, a channel type based on depth information

To be able to use depth information for classification tasks, the data has to be transformed into alternate representations, similar to the process described in 3.1.
A common idea is to directly apply the idea of HOG, which is known to give good results, to the new data type. Spinello and Arras [21] proposed a detector based on histograms of oriented depths (HOD), as well as a detector Combo-HOD that combined the outputs from HOG and HOD. Their algorithm was tested using data from a Kinect sensor gathered indoors. They compared their results with detectors based only on visual data and were able to achieve improved performance compared to them. The test data was biased towards not working well for detectors based purely on vision data, as the scenes are not illuminated well. This means that while their results are not meaningful to determine how Combo-HOD improves detection in good conditions, it is clear that it results in better performance in difficult conditions
Premebia *et al* [18] followed a similar approach of using HOG on depth data. However for testing they used the LiDAR depth data from the KITTI dataset, which is taken in an

urban environment. Their results also point out that combining depth with visual information improves overall detection accuracy.
The image representation of HOD is very similar to the other channels ICF uses. Therefore adding it to the detection algorithm would not be difficult. The findings mentioned in this section indicate that extending ICF in this way will improve its detection accuracy, especially in difficult lighting conditions.

7.3.2 Improving detection speed

Depth data can have other uses than just improving detection accuracy. The information encoded in depth maps can be used to speed up detection.
As described previously, many detection algorithms are based on a sliding window and a pyramid of scales. Beforehand it is not known what sizes of pedestrians there are in an image, therefore the pyramid has to be finely sampled to not miss detections. However the size of pedestrians in an image is largely dependent on their distance from the camera. Pedestrians that are farther away appear smaller, while pedestrians close by are large. This means that by considering average human height, the number of scales that are scanned for pedestrians can be reduced. If a scale corresponding to a certain pedestrian height does not contain depth data of a calculated value, the scale does not have to be checked, making detection faster. This concept is called informed scale-space search and was proposed by Spinello and Arras [21].
Even though ICF tries to alleviate the bottleneck of scale-space detection by approximating most of the scales, adding informed scale-space search would result in an improvement in terms of speed.

7.3.3 Introducing redundancy to improve safety

Accuracy is a major concern in object detection, and especially in pedestrian detection, as missed detection can lead to severe consequences. Using detection algorithms in real-world settings, such as for driving assistance or mobile robotics, requires the programs to be very accurate. A different approach to simply adding depth data channels such as HOD to a detector is to train two detectors and run them in parallel. One detector relies only on depth data while the other evaluates vision data. This way, the depth detector is complementary and redundant and serves to ensure that classification results are correct. In addition to that, should one system fail the other can still function with good precision. This will improve safety and allow pedestrian detection to be applied in more use cases.
For this idea to work, an accurate classification algorithm that only makes use of depth data has to exist. Premebida *et al* considered this problem in their work on pedestrian detection using LiDAR data [19]. They implemented a number of detection algorithms and tested them on datasets that they collected themselves. With a detection rate of 90% at about $5 \cdot 10^{-2}$ fppi the algorithms performed worse than state-of-the-art vision-based detectors (the previously described ICF detector achieves 92% detection at 10^{-4} fppi).

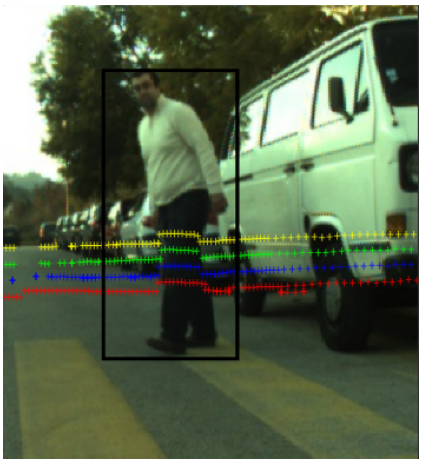


Figure 7.2: The depth data used by the algorithms in [19]

However the accuracy is still good enough to work complementary to a vision-based classifier and still produce decent results should a system failure occur. In addition to that, the performance of entirely depth-based systems can most likely be improved, because as shown in figure 7.2, the algorithms tested by Premebida *et al* make use of only a small part of the information potentially available.

8 Conclusion

The ICF detector

Pedestrian detection in urban environment is a topic that is very important for future technologies, especially the self-driving car. The ICF detection algorithm is very capable of producing high detection rates with a low number of false positives. In the design of this algorithm a lot of thought also went into how a good execution time can be achieved, which is critical for real-time applications.

AdaBoost

AdaBoost is a good alternative to SVMs for training. It is very fast, even when working with large amounts of training data, and can easily be sped up by multithreading. This makes it very suitable for running on modern CPUs and GPUs.

Depth detection

Augmenting vision-based detection with depth data is a more recent idea, however there already are a number of promising results by multiple researchers. This indicates that further research will enable decisive improvements of the current state-of-the-art in detection.
Depth data will improve the accuracy of classifiers by adding more informative channels that can be evaluated. It will also enable a speed-up of detection algorithms, as the information can be used to find regions of interest. In use cases where safety is a concern, depth data detection can be used as a redundant system to avoid classification errors and their consequences.

Future work

The version of the ICF detection algorithm that I implemented can be greatly improved in terms of execution time. Constructing multiple scales for an image pyramid and classifying a large number of windows on each of them are independent tasks that are very suitable for multithreading. My implementation utilizes only a single core of a CPU, a parallelized version would be able to make use of much more. It even makes sense for the algorithm to run on a GPU, because the number of windows classified on each frame ranges in the millions. A large speed-up could therefore be achieved by an OpenCL or a CUDA implementation.

To a lesser extent, the same is true for the training process. Training on a system with more RAM would also mean that it would be possible to evaluate all features only once and then saving the results, which further improve run-time. Additionally, the accuracy of the detection step could be improved by spending more time optimizing the parameters for training. Some ideas for this are varying the depth of the decision trees used as the weak learners and using a larger dataset that contains more varied positive and negative images for training. If training time is reduced, it would also be reasonable to use all available features for training, instead of only a randomly picked subset.
The ideas presented in 7 are not implemented in my program, as no dataset with easily usable depth data was available. By preprocessing the data from the KITTI or ETH-Zurich dataset, my propositions for augmenting detection with depth data could be tested in combination with the ICF detector. This would be valuable research, because the ideas have so far only been tested with inferior datasets and comparison has only been done with older detectors that have a worse performance than ICF.

Appendix

Bibliography

[1] 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.

[2] 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.

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

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

[5] Piotr Dollár, Zhuowen Tu, Pietro Perona, and Serge Belongie. Integral channel features–addendum. *vision. ucsd. edu*.

[6] Piotr Dollár, Zhuowen Tu, Pietro Perona, and Serge Belongie. Integral channel features. 2009.

[7] Piotr Dollar, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE transactions on pattern analysis and machine intelligence*, 34(4):743–761, 2012.

[8] Markus Enzweiler and Dariu M Gavrila. Monocular pedestrian detection: Survey and experiments. *IEEE transactions on pattern analysis and machine intelligence*, 31(12):2179–2195, 2009.

[9] A. Ess, B. Leibe, and L. Van Gool. Depth and appearance for mobile scene analysis. In *International Conference on Computer Vision (ICCV'07)*, October 2007.

[10] A. Ess, B. Leibe, K. Schindler, , and L. van Gool. A mobile vision system for robust multi-person tracking. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'08)*. IEEE Press, June 2008.

[11] 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.

[12] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

[13] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[14] Laura Leal-Taixé, Anton Milan, Ian Reid, Stefan Roth, and Konrad Schindler. Motchallenge 2015: Towards a benchmark for multi-target tracking. *arXiv preprint arXiv:1504.01942*, 2015.

[15] Matthias Luber, Luciano Spinello, and Kai O Arras. People tracking in rgb-d data with on-line boosted target models. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 3844–3849. IEEE, 2011.

[16] Anton Milan, Laura Leal-Taixé, Ian Reid, Stefan Roth, and Konrad Schindler. Mot16: A benchmark for multi-object tracking. *arXiv preprint arXiv:1603.00831*, 2016.

[17] Constantine Papageorgiou and Tomaso Poggio. A trainable system for object detection. *International Journal of Computer Vision*, 38(1):15–33, 2000.

[18] Cristiano Premebida, Joao Carreira, Jorge Batista, and Urbano Nunes. Pedestrian detection combining rgb and dense lidar data. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 4112–4117. IEEE, 2014.

[19] Cristiano Premebida, Oswaldo Ludwig, and Urbano Nunes. Exploiting lidar-based features on pedestrian detection in urban scenarios. In *Intelligent Transportation Systems, 2009. ITSC'09. 12th International IEEE Conference on*, pages 1–6. IEEE, 2009.

[20] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[21] Luciano Spinello and Kai O Arras. People detection in rgb-d data. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 3838–3843. IEEE, 2011.

[22] 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–I. IEEE, 2001.

[23] Christian Wojek, Stefan Walk, and Bernt Schiele. Multi-cue onboard pedestrian detection. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 794–801. IEEE, 2009.