

Computer vision for pedestrian detection using Histograms of Oriented Gradients



Jose Marcos Rodríguez Fernández
Facultat Informàtica de Barcelona
Universitat Politècnica de Catalunya

A thesis submitted for the degree of
Engineer in computer science

2014 January

1. Vocal: Francisco Javier Larrosa Bondia

2. Secretary: Joan Climent Vilaró

2. President: Manel Frigola Bourlon

Day of the defense: 31/03/2014

Abstract

This work targets the pedestrian detection in static images from a computer vision point of view. The interest of such detector resides in its many applications; automotive safety, crowd control, video surveillance or automatic image indexing are just a few examples. Detecting pedestrians is a challenging matter as persons can adopt a wide range of poses, in very different backgrounds and under significant changes in illumination and color. To achieve a robust detection method we study and develop a *HOG* plus *SVM* solution, as proposed by Dalal & Triggs. The *HOG* descriptor proposed turns out to be robust to small changes in the image contour, location and direction, and significant changes in illumination and color. Even though *HOGs* perform equally well for other classes, in this work we target specifically in upright persons, or to say, pedestrians. Furthermore we try several *SVM* models and training approaches to pick out the best possible *SVM* kernel and parameters for two different well known persons data sets; MIT and INRIA data sets.

Dedicated to my mother for her tireless efforts and kind soul.

Contents

List of Figures	vii
List of Tables	xi
Glossary	xiii
1 Introduction	1
1.1 Computer vision	1
1.2 Some background on object detection	3
1.2.1 Challenges in modeling the object class	4
1.2.2 Challenges in modeling the non-object class	5
2 Aims of the project	7
2.1 Final aim	7
2.2 Preliminary aims	8
3 A robust feature for object recognition	9
3.1 Introduction	9
3.2 HOG configuration	9
3.3 Extracting HOGs from an image	11
3.3.1 Brief overview of HOG descriptors	11
3.3.2 Input Images	12
3.3.2.1 Image size	12
3.3.2.2 Color spaces	12
3.3.3 Gradient computation	13
3.3.4 Spatial / Orientation Binning	14
3.3.5 Normalization and descriptor blocks	15

CONTENTS

3.4	Visualizing the data	16
4	SVM	21
4.1	What is SVM?	21
4.2	How it works?	21
4.2.1	SVM kernels	23
4.2.2	Training a SVM	26
4.2.2.1	Selecting the SVM parameters	28
4.2.3	Testing a SVM	31
5	Pedestrian detection	33
5.1	Training the detector	33
5.1.1	Data sets	34
5.1.2	SVM models	36
5.1.2.1	MIT models	36
5.1.2.2	INRIA models	37
5.2	Testing the detector	44
5.3	The final detector	47
6	Implementation and Usage	51
6.1	Implementation	52
6.1.1	Auxiliar functions	56
6.2	Usage	57
7	Performance	59
7.1	MIT models	59
7.2	INRIA models	61
7.2.1	1st training approach	62
7.2.1.1	Linear models	62
7.2.1.2	RBF models	64
7.2.1.3	Linear versus RBF model comparison	66
7.2.2	2nd training approach	68
7.2.2.1	Linear models	68
7.2.2.2	RBF models	68
7.2.2.3	Linear versus RBF model comparison	69

7.2.3	Selecting the final model	69
7.3	Detector performance	72
8	Discussion	75
8.1	Possible improvements	75
8.1.1	Code compilation.	75
8.1.2	Parallel GPGPU computation.	76
8.1.3	Principal Component Analysis for dimensional reduction.	77
8.2	Alternatives	79
9	Materials & methods	81
9.1	MATLAB	81
9.1.1	variable types	81
9.2	Libraries and packages	82
9.2.1	libSVM	82
9.2.2	NIST DET plots	82
9.2.3	m2html	82
10	Project Management	85
10.1	Planning	85
10.2	Costs	89
10.2.1	Human resources	89
10.2.2	Hardware resources	89
10.2.3	Software cost	90
	References	93

CONTENTS

List of Figures

1.1	Relations between computer vision and many other fields	2
3.1	Cell and block size performance impact	10
3.2	Descriptor size equation	10
3.3	HOG extraction general view	11
3.4	Gradient equations	13
3.5	Gradient magnitude equation	13
3.6	Gradient angle equation	14
3.7	Bin k equation	15
3.8	Linear interpolation equation	15
3.9	L1-normalization for the v descriptor vector of a block	16
3.10	L1-squared-normalization for the v descriptor vector of a block	16
3.11	L2-normalization for the v descriptor vector of a block	16
3.12	HOG t-sne map	18
3.13	HOG common representation	18
3.14	HOG HOGgles representation	19
4.1	SVM maximum separating hyperplane margin	22
4.2	Different hyperplanes in 2D	23
4.3	Graphic examples of 2D non linear separable data.	24
4.4	Feature mapping to a higher dimensional space	25
4.5	Gaussian kernel formula.	25
4.6	Cost parameter effect	27
4.7	Gaussian curves as similarity functions	28
4.8	γ effect on the classification boundary	28

LIST OF FIGURES

4.9	Learning curve showing the evolution of the error in function of the model complexity	30
4.10	Typical learning curve for high variance	30
4.11	Typical learning curve for high bias	31
5.1	Scale-space pyramid illustration	34
5.2	MIT pre linear model cross-validation	36
5.3	MIT round 1 linear model cross-validation	37
5.4	Cross validation accuracy tendency round comparison	39
5.5	INRIA pre linear model cross-validation using all the negative training instances	39
5.6	INRIA linear model round 1 cross-validation	40
5.7	RBF parameter cross validation search (1st approach)	42
5.8	First RBF parameter search	43
5.9	RBF parameter cross validation search (2nd approach)	43
5.10	precision and recall equations	45
5.11	F-score equation	45
5.12	Multi detection versus non-max-suppression detection	49
6.1	Static-detector compiled version	57
7.1	MIT linear def model threshold search	61
7.2	Color information impact in MIT linear model	61
7.3	INRIA linear rgb pre-model versus round-1 model DET curve (1st approach)	63
7.4	re-train impact in INRIA linear model (1st approach)	63
7.5	INRIA linear rgb round1 model threshold search (1st approach)	64
7.6	INRIA RBF rgb training rounds DET curves (1st approach)	65
7.7	re-train impact in INRIA RBF models	65
7.8	optimal threshold search for INRIA RBF models (1st approach)	66
7.9	INRIA linear vs rbf models DET curves (1st approach)	67
7.10	INRIA linear model versus RBF model performance comparison (1st approach)	67
7.11	INRIA linear rgb per-model versus round-1 model	68

LIST OF FIGURES

7.12	Re-train impact in INRIA linear model (2nd approach)	69
7.13	INRIA RBF model performance due to the parameter configuration . .	70
7.15	Linear models comparison between the 1st and 2nd training approach .	70
7.14	INRIA linear versus rbf model performance comparison (2nd approach)	71
7.16	INRIA RBF model comparison between 1st and 2nd training approach .	72
7.17	Detector time in seconds w.r.t the number of evaluated windows	73
7.18	MATLAB code analysis for C code generation	74
8.1	GPU implementation steps followed in (1)	76
8.2	INRIA linear PCA-model cross-validation curve	78
8.3	Hidden layer feature representation learned by a simple ANN	80
9.1	<i>MATLAB</i> matrix data types	81
10.1	Overall project schedule	85
10.2	Research schedule	86
10.3	Implementation schedule	87
10.4	Building schedule	87
10.5	Building schedule	88
10.6	Complete schedule	88

LIST OF FIGURES

List of Tables

7.1	MIT linear models performance comparison table	60
7.2	INRIA RBF final model performance.	69
8.1	INRIA linear PCA-model performance.	78
10.1	Human resources cost breakdown	89
10.2	Software costs	90
10.3	Total project cost	91

GLOSSARY

Glossary

ANN	Artificial Neural Network	HOG	Histogram of Oriented Gradients
CNN	Convolutional Neural Network	MIT	Massachusetts Institute of Technology
DET	Detection Error Trade off	NIST	National Institute of Standards and Technology
FLOPS	FLoating-point Operations Per Second	RAM	Random Access Memory
GPGPU	General-Purpose GPU	RBF	Radial Basis Function
		ROC	Receiver Operating Characteristic
		SIFT	Scale-Invariant Feature Transform
		SVM	Support Vector Machine
		t-SNE	t-distributed Stochastic Neighbor Embedding

GLOSSARY

1

Introduction

Computers and machines are ubiquitous elements in our daily lives and this tends to be an increasing reality day by day. Its natural to try to improve the computer-human or computer-environment interaction. Even though computers process and treat huge amounts of data with an ease that humans have failed to achieve, human daily problems like recognizing another person or obtaining high level information from a scene turns out to be something machines do not deal with the same ease as human do. This fact was somehow evidenced in the 80s by *Hans Moravec* who pointed out that human reasoning needs relatively little computation while sensorimotor skills require enormous amounts of computation.¹ Therefore one of the most challenging points of the modern engineering is to achieve a good interaction between machines and their environment. This requires computers to have some intelligence and learning capability. Specially speech recognition and visual interpretation are key points to grant computers and machines with good interaction skills.

This chapter presents a brief description of the computer vision field, some background of it and the main challenges it presents.

1.1 Computer vision

Computer vision or artificial vision is a field that comprehend methods, techniques and technologies to acquire, process and understand images. Typical targets in computer

¹As Moravec writes: "It is comparatively easy to make computers exhibit adult level performance on intelligence tests or playing checkers, and difficult or impossible to give them the skills of a one-year-old when it comes to perception and mobility."

1. INTRODUCTION

vision include;

- detection, segmentation and localization of objects (i.e face identification)
- object tracking (i.e following pedestrian movements for surveillance)
- image search by its content (i.e image online searchers)
- image restoration (i.e noise or motion blur removal)
- 3D construction from a set of 2D images

Even though many computer vision applications are programmed to solve a particular problem, it is becoming common to find method based on machine learning. Therefore computer vision is highly related with other fields. Figure 1.1 shows this relations.

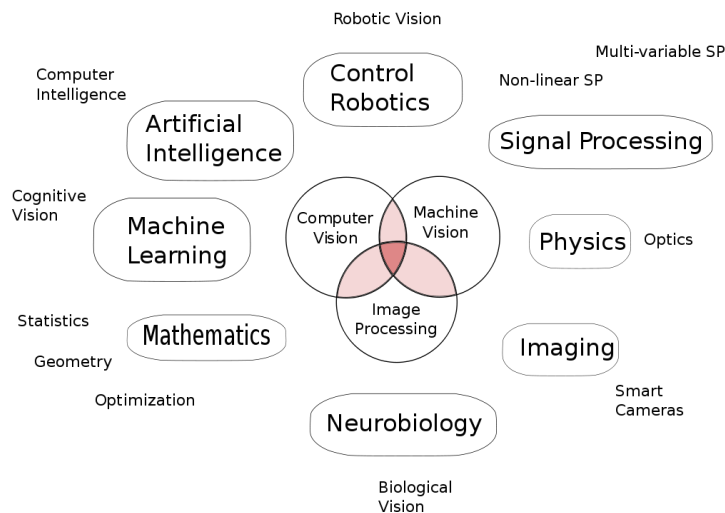


Figure 1.1: Relations between computer vision and many other fields -

As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. In a more technological point of view, among its applications we could highlight:

- Industrial quality control (i.e detecting cracks or fissures in tiles, bottles etc)
- Autonomous driving
- Detecting events in video surveillance

- Indexing image data bases
- Modeling object or environments (i.e medical image analysis)
- Computer-human interaction (i.e controlling a computer with the eye movement)
- Automatic document treatment (i.e bank checks verification)

1.2 Some background on object detection

In this thesis we target in searching objects, in this case, pedestrians within static images. Mainly an object detection application can be splitted in two different stages; an encoding stage where the image or a region of it is translated into a feature vector representation, and a detection stage to decide whether some concrete region is or not a positive match. The feature extraction representation can either be sparse or dense and usually capture shapes, contour information or intensity patterns. The sparse feature approach extracts information only from some special regions, assuming not the same importance over the whole image. Its noteworthy that this method needs some way to detect important regions what implies a dense search over the image. Nevertheless the way this points are found differs from the final encoding of the dense feature approach. Additionally the space relationships between the features representation can also be used to make detection decision. From the detector point of view mainly two approaches can be found. On one hand a part based approach, where a detection is defined by the presence of different parts constituting the object to detect and the spacial relations between these parts. Considering the pedestrian detection problem we could think on a pedestrian like a set of arms, legs, head and torso. For this set of parts to effectively be a person it is not enough to find each part, it is also needed to satisfy some spatial ordering. On the other hand a kind of template approach, where a densely feature is computed to, further on, classify it as matching or not by any learning algorithm, commonly *SVM* or similar techniques.

Among the pedestrian detection proposed solutions and methods, one of the most promising approaches seems to be the sliding window paradigm when dealing with low or medium resolution images.

1. INTRODUCTION

Papageorgiou et al. proposed one of the first sliding window plus *SVM* with an over-complete dictionary of multiscale Haar wavelets (2). Viola and Jones built their detector upon integral images for fast feature computation and a cascade structure for efficient detection, and utilizing AdaBoost for automatic feature selection (3). Zhu et al. sped up HOG features by using integral histograms (4, 5). Dollar et al. proposed an extension of Viola and Jones method where Haar-like features are computed over multiple channels of visual data (LUV color channels, grayscale, gradient magnitude and gradient magnitude quantized by orientation) (6). The *Fastest Pedestrian Detector in the West (FPDW)*, extended this approach to fast multi-scale detection after it was demonstrated how feature computed at a single scale can be used to approximate feature at nearby scales. Achieving a 5fps on 640×480 images as this method does not need to compute a fine scale-space-pyramid. (7).

In this thesis we densely compute our feature vector representation over image regions following a template approach, and then classify every feature with a *SVM*. The goal is to model pedestrians in such a way that light, background or their clothes does not detriment the detection. Concretely this point leads us to briefly review some of the most common issues that object-detection problems present. These can be divided in two categories, on one hand we can talk about the ones encountered when modeling the object class that we want to detect, on the other hand, the ones in modeling the non-object class.

1.2.1 Challenges in modeling the object class

Among the several difficulties that make hard modeling and detecting a particular object we could mention the following:

- *Illumination variations*: objects appear widely different and modeling them in such a way they become invariant to this variations turns out to be a key point.
- *Object pose or object deformation*: changes what an object looks like and therefore template approaches for object recognition tend to suffer low performance in high changing objects.
- *Cluttered scenes*: hinder detection and localization of objects.

1.2 Some background on object detection

- *Occlusion*: makes hard to recognize shapes and therefore objects as some parts can not be seen.
- *Intra-class appearance*: makes difficult to group dissimilar objects. (i.e car types, colors, shapes widely vary but they all represent cars)
- *Viewpoint*: makes appearance to widely vary (i.e pedestrians from an aerial point of view versus frontal view)

1.2.2 Challenges in modeling the non-object class

On the other hand for the non-object class we should mention:

- *Bad localization*: sub-parts of the object could be detected but not the whole object.
- *Confusion with similar objects*
- *Miscellaneous backgrounds*

1. INTRODUCTION

2

Aims of the project

2.1 Final aim

The main objective of the thesis is to implement and evaluate a pedestrian detection method for static images. We relay in the descriptor proposed by Dalal and Triggs, from the French National Institute for Research in Computer Science and Control (INRIA). Their solution consists in a HOG plus a SVM approach which we build using the MATLAB language and environment. More specifically we compare the effectiveness of different set of parameters, SVM models, kernels, pedestrian data sets, and variations of the detection method, while achieving a completely understanding of Support Vector Machines, kernels and the HOG descriptor itself. Furthermore we realize a complete set of tests and measures to each trained detector to prove and show in different manners which turns out to be the best possible configuration taking into account detection accuracy in terms of a trade-off between two different types of errors. In addition every bit of code is developed with flexibility and abstraction in mind so every experiment and model realized in this thesis can be replicated with same or different parameters, configurable through parameter files. This enables other people to find extra results or make different comparisons. For the same reason a wide set of routines and functions are also available to provide a way to produce the same data results and make comparisons easier.

2. AIMS OF THE PROJECT

2.2 Preliminary aims

Before tackling the problem some research was made to have a better understanding of the pedestrian detection problem, its applications and how other people approached the solution. Besides, in order to accomplish the above mentioned objective some familiarization with *SVM*, its details and difficulties that may appear when trying to shape a solution to this kind of computer vision problems is required. As well as with various testing, accuracy measures and methods in order to make a complete comparison. Concretely a thorough study of the mathematical basis of *SVM* and its kernels, together with a study of which problems and how to diagnose and solve them, when dealing with machine learning problems, has been done. Moreover some high dimensional reduction and visualization techniques has been explored to find some procedures that permit to examine easier how the studied *HOG* descriptor characterizes the desired data. Finally before starting any implementation a brief research and comparison was done in order to determine which environment or language was more appropriate to carry out the whole project development, taking into account available tools or libraries, which requires less effort or lines of code or which had a more active community developing similar tools and applications just to name a few aspects.

3

A robust feature for object recognition

3.1 Introduction

The main goal of this chapter is to explain an outperforming descriptor for object recognition. Both from a theoretically point of view as from a more pragmatic and computationally point of view. As has been said in the previous sections of this thesis the real challenge is finding a descriptor that performs equally well independently of illumination variations and within a relatively wide range of poses variations that can be adopted by pedestrians, this is a descriptor able to cleanly discriminate between pedestrians and not pedestrian even in cluttered background in difficult illumination environments. After examining all this requirements Dalal and Triggs (8) end up with a feature called *Histogram of Oriented Gradients* a reminiscent of edge orientation histograms(9, 10), SIFT descriptors (*Scale- Invariant Feature Transform*)(11) and shape contexts.(12).

Even though this work is completely focused on their usefulness regarding pedestrian detection it is worth mentioning that *HOGs* performs equally well in other shape-based object classes, showing substantial gains over intensity based features.

3.2 HOG configuration

For this thesis the default HOG configuration used by Dalal and Triggs in (8) has been used. The reader may notice that the configuration used in this work differ slightly

3. A ROBUST FEATURE FOR OBJECT RECOGNITION

from the best one found in Dalal and Triggs in their work. This fact has two main reasons to be. The first one is to successfully compare the models and experiments performed in this work with the ones developed by them. The second one is that the best configuration choice increments the size of the descriptor as having more redundant information.

The selected configuration consists of linear gradient voting into 9 orientation bins in 0° to 180° ; 16×16 pixel blocks of four 8×8 pixel cells. Figure 3.1 shows the performance depending on the configuration choice.

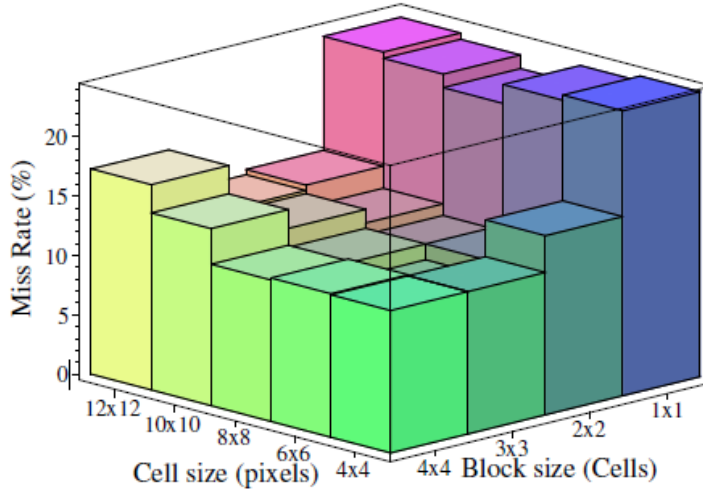


Figure 3.1: Cell and block size performance impact - The miss rate at 10^{-4} FPPW as the cell and block sizes change. The stride (block overlap) is fixed at half of the block size. 3×3 blocks of 6×6 pixel cells perform best, with 10.4% miss rate.

As the performance of the selected configuration is pretty much the same as for the best one lets compare the descriptor size. For the default configuration the descriptor is a vector of 3780 components but for the optimum configuration is a 12312 component vector. In general the size of the descriptor can be calculated as:

$$\text{Descitor size} = b_s^2 \cdot n_b \cdot (v_{cells} - b_s + \sigma) \cdot (h_{cells} - b_s + \sigma) \quad (3.1)$$

Figure 3.2: Descriptor size equation

(n_b : number of bins, b_s : block size, σ : stride between blocks)

3.3 Extracting HOGs from an image

Lets review now a complete definition of the descriptor, the key points of its obtainment and the main configuration used in this thesis.

3.3.1 Brief overview of HOG descriptors

In the introduction of this chapter other descriptors were said to be similar as the one we are studying, the main difference is that HOGs are computed on a dense and overlapping grid of equispaced cells.

The main idea behind HOGs is that the local appearance and shape of an object in an image can be characterized as the intensity gradient distribution, this is, by the edges directions. This characterization can be accomplished by dividing the image in little and connected regions, named cells, from within the gradients are computed. Roughly speaking to make the descriptor we should join all the gradients computed through all this regions and count the occurrences of the orientations. Anyway further improvement can be achieved by defining a larger area, called block, where cells are grouped. This blocks are then used to compute a intensity measure of the whole block, used no normalize all the cells within the blocks. Proceeding like this makes the descriptor more invariant to illuminations changes.

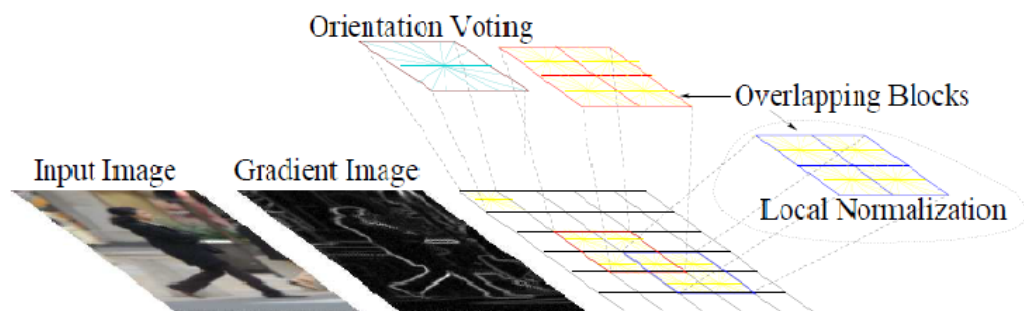


Figure 3.3: HOG extraction general view - graphical description of all the extraction steps

3. A ROBUST FEATURE FOR OBJECT RECOGNITION

3.3.2 Input Images

3.3.2.1 Image size

All the images used in this work, and therefore the detection window, are 64×128 . Dalal & Triggs spotted out that shorter window size with less margin between the pedestrian and the image borders turned out in a loss of performance. Therefore every window present a margin of about 16 pixels from the pedestrian to all four sides. Reducing this margin to only 8 pixels (48×112 window) decreases performance by 6%. Increasing the pedestrian size within a 64×128 window also causes a similar loss of performance as the margin results decreased even though the person resolution is increased. The information in these margins provides useful context information that helps the detection.

3.3.2.2 Color spaces

A color model is an abstract mathematical model describing the way colors can be represented as tuples of numbers, typically as three or four values or color components. For the present case, we are specially interested in the *RGB* model as is the color space in which our images are encoded. Its name comes from the initials of each of the three additive primary colors (*Red, Green and Blue*), as it is an additive color model, specially designed for representing and displaying images in electronic systems, anyway before its electronic usage was a solid model based on the human color perception theory. Even though the *RGB* model has several implementations, the most common one uses 8 bits per channel, giving a total of 24 bits to encode a concrete color, or equivalently $256^3 \simeq 16.7$ colors.

We are specially, but not only, interested in the *RGB* color space as we are also dealing with gray-scale images. While in the *RGB* model each pixel was represented by 24 bits (8 bits per channel), in the gray-scale model every pixel is commonly encoded with only 8 bits, therefore every pixel value ranges from 0 up to 255, giving 256 gray levels. As we have said, an input image can be in different pixel representations, so this thesis has evaluated two different options; Gray-scale images and *RGB* images.

3.3.3 Gradient computation

Even though many feature computations require some kind of image normalization, such as color or gamma normalization, this method does not need to pre-process the image due to the subsequent steps of the descriptor computation, therefore the first step is to calculate the gradients. Dalal and Triggs (8) evaluated several derivative masks, both 2-D and 1-D masks, and they found that the performance was quite sensitive to the way this gradients were computed.

Within the 2-D derivative masks we find 2x2 diagonal masks $\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$, $\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$ and Sobel masks (13)

And from the 1-D alternatives we find; centered: $([-1, 0, 1])$, uncentered: $([-1, 1])$ and cubic corrected: $([1, -8, 0, 8, 1])$ masks.

Also several gaussian smoothing (14) scales were tested, including no smoothing. It turned out that no smoothing and the simplest 1-D centered derivative mask performed better than all the others.

Therefore in this thesis we center in the evaluation of the best choice found by (8), so the gradient is computed applying $[-1, 0, 1]$ for the horizontal direction and $[-1, 0, 1]^T$ for the vertical direction.

By applying a mask we refer to calculate the convolution of the masks as they go through the image, so at every pixel we calculate a value for the x-derivative and another value for the y-derivative, this values correspond to the x and y gradient magnitudes respectively, lets name them G_x and G_y .

The equations defining the gradients are, respectively (being I the image and (i, j) the pixel coordinates):

$$G_x(i, j) = \frac{\partial I}{\partial x}(i, j) \quad (3.2)$$

$$G_y(i, j) = \frac{\partial I}{\partial y}(i, j) \quad (3.3)$$

Figure 3.4: Gradient equations

The gradient magnitude itself is then computed as the square root of the quadratic sum of each gradient component, this is:

$$M(i, j) = \sqrt{G_x^2(i, j) + G_y^2(i, j)} \quad (3.4)$$

3. A ROBUST FEATURE FOR OBJECT RECOGNITION

Figure 3.5: Gradient magnitude equation

As regards the angle it can be calculated as the four-quadrant inverse tangent of G_y and G_x , namely:

$$\Theta(i, j) = \arctan\left(\frac{G_y(i, j)}{G_x(i, j)}\right) \quad (3.5)$$

Figure 3.6: Gradient angle equation

However we have not commented yet two details about the HOG obtainment. The first detail concerns the already mentioned aspect of the color space of the input image. The process obtained so far does not mention how to deal with image colors, so for the RGB image case. As we will see in §7 using when available the three channels improves slightly the overall performance. The basic idea behind the HOG computation when taking a three channel image is mainly apply the explained method independently for all the channels and taking for each pixel of the image the gradient with highest value and its correspondent angle.

The second detail has to do with the boundary condition, that is to say how to compute the gradient at the image boundaries where the center of the derivative mask can not be placed as it should. Anyway we can forget this detail by now, just to not obscure the main idea behind the descriptor itself, this will be explained in more detail when we review the concrete implementation in §6.

3.3.4 Spatial / Orientation Binning

This step introduces the non linearity of the descriptor and implies the creation of an histogram for a bunch of local spatial areas, the ones that we called *cells*. If we remember the first and brief definition given in the introduction, the image was subdivided in little cells. Thus for every pixel within a cell a weighted vote is issued for the bin corresponding to the angle of its gradient, then for all the pixels of a cell the issued votes are accumulated to form the final histogram for that cell. This histograms represent angles evenly spaced between 0° and 180° or within 0° and -360° , depending whether the angle is signed or unsigned respectively.

The authors found out that using unsigned angles together with 9 orientation bins, was the outperforming configuration. Probably the reason why omitting angle sign information is better is because in human detection we find a wide range of clothing and background colors and therefor this information is uninformative.

3.3 Extracting HOGs from an image

With respect to the vote weighting either squared gradient magnitude, the square root or the magnitude itself could be an option. In practice using the magnitude itself results to be the best option.

Therefore the resulting equation for the computation of the k^{th} bin of the histogram is:

$$h_k = \sum_{i,j} M(i,j)1[\Phi(i,j) = k] \quad (3.6)$$

Figure 3.7: Bin k equation

(1 is the characteristic function that indicates if a particular orientation belongs to a given bin or not)

In our concrete case, k ranges from 1 to 9. As Dalal and Triggs showed in (8), a fine orientation coding turns out to be essential while a more coarse spatial coding can be done.

In fact, given the previous angle division in bins, probably given an angle it will fall between to bin centers, so the vote is calculated via a bilinear interpolation between the two neighboring bin centers. Doing the voting like this reduces the aliasing. The bilinear interpolation is done as a double linear interpolation in both orientation and position.

$$f(x|x_1, x_2) = f(x_1) + \frac{f(x_2) - f(x_1)}{x_2 - x_1}(x - x_1) \quad (3.7)$$

Figure 3.8: Linear interpolation equation

Summarizing we have that the image is by now divided in little sub-regions called cells, at the same time cells are grouped in larger spatial areas called blocks. Then for every pixel within a cell the gradient ($G(i,j)$) is computed and a vote is issued, this vote is then linearly interpolated to discover which portions of its magnitude ($M(i,j)$) correspond to each contiguous bin. So at the same time every pixel contributes to a couple of bins and at each of the four histograms that form a block.

3.3.5 Normalization and descriptor blocks

As has been said over all the whole work, detecting pedestrian is subject to the main computer vision issues like illumination and background variations or occlusions. For the sake of a robust descriptor some kind of illumination normalization must be done,

3. A ROBUST FEATURE FOR OBJECT RECOGNITION

this is evidenced when paying attention to the variations in the gradients strength. Actually this step is essential to achieve good results.

Several normalization schemes were explored by (8). Lets define first v as the vector containing all the histograms for a given block, $\|v\|_k$ the k-norm of v , with $k \in 1, 2$ and lets ϵ be a small constant. Then the normalization schemes are:

$$\text{L1-norm : } v \rightarrow \frac{v}{\|v\|_2 + \epsilon} \quad (3.8)$$

Figure 3.9: L1-normalization for the v descriptor vector of a block

$$\text{L1-squared norm : } v \rightarrow \sqrt{\frac{v}{\|v\|_2 + \epsilon}} \quad (3.9)$$

Figure 3.10: L1-squared-normalization for the v descriptor vector of a block

$$\text{L2-norm : } v \rightarrow \frac{v}{\sqrt{\|v\|_2^2 + \epsilon^2}} \quad (3.10)$$

Figure 3.11: L2-normalization for the v descriptor vector of a block

And L2-norm followed by a clipping, limiting the values of v to 0.2 and re-normalizing again. This can be achieved by getting the result of the L2-norm and cutting it and normalizing again.

The experiments showed that either L1-norm-squared, L2-norm or L2-norm-Hys performs similar and achieves good results, but L1-norm decreases performance in a 5%. Not normalizing penalizes enormously the performance in around a 27%.

One could suppose that including the ϵ value in the above mentioned calculations may in any manner disrupt or distort the discrimination defectiveness of the descriptor but the results are insensitive to a wide range of ϵ values.

3.4 Visualizing the data

In general before dealing with any classification task, sometimes helps visualizing how the selected descriptor represents the elements we want to classify. Obviously when the dimension of the descriptor increases more difficult becomes the task of visualizing how the selected features represent the data and therefor more difficult to determine at a

glance if that set of features in conjunction with a concrete classification method will lead to success or will fail.

In this case we are trying to visualize points on a 3780 space as the explained HOG configuration gives a such dimensional vector. Once assumed this space is impossible to really be visualized, the only possible way to figure out how the data is distributed is by applying some dimensional reduction method. The idea is to end up with a 2D or 3D plot revealing in some manner the implicit relations between points. A good descriptor plus a good high visualization technique should give a map where instances of the same class are close in the map and different classes are far from each other.

We selected t-distributed Neighbor Embedding (*t-SNE*) (15) method for achieving a visualization from HOGs representation. The reason is because of the improvement that this method presents compared with other similar ones. Plots from t-SNE present a lower tendency to crowd points together in the center of the map and is better in representing structures at many different scales in a single map.

Before presenting the results of the t-SNE visualization a very brief explanation of how it works follows. Two stages comprise the algorithm. First, t-SNE constructs a probability distribution over pairs of high-dimensional objects in such a way that similar objects have a high probability of being picked, whilst dissimilar points have an infinitesimal probability of being picked. Second, t-SNE defines a similar probability distribution over the points in the low-dimensional map, and it minimizes the Kullback-Leibler (16), divergence between the two distributions with respect to the locations of the points in the map.

Looking at the visualization provided by the t-sne algorithm one can observe at least a strong relationship between points representing HOGs extracted from pedestrian windows and the ones extracted from windows not containing any pedestrian.

Once we have some clue about the discrimination capabilities of the HOG descriptor we can start thinking about classification methods that we will explain in detail in the next chapter.

Moreover, instead of visualizing the whole bunch of points representing every instance, we may want to visualize a HOG feature extracted from a concrete image, is to say, to visualize the image in the feature space, something similar to view the same as the classifier system sees. One possible approach to directly plot the values of each histogram bar for all cells in the image, commonly represented in a star shape. Figure 3.13 shows

3. A ROBUST FEATURE FOR OBJECT RECOGNITION

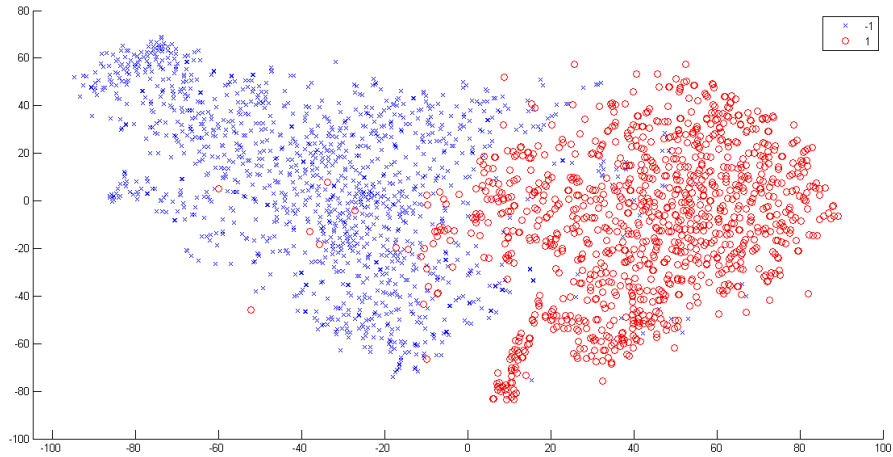
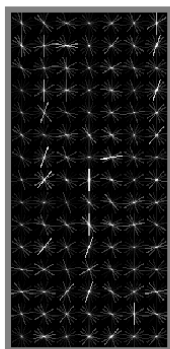


Figure 3.12: HOG t-sne map - red circles represent pedestrians, blue crosses not pedestrians

some examples of this representation. Another approach, more complex but quite better to understand why our detector fails in some cases, is the method proposed by Carl Vondrick, Aditya Khosla, Tomasz Malisiewicz and Antonio Torralba from *MIT* (17). Figure 3.14 show some examples in image and feature space, the HOGgles representation show how the HOG descriptor seems to belong to person images.



(a) HOG representation



(b) Original image

Figure 3.13: HOG common representation

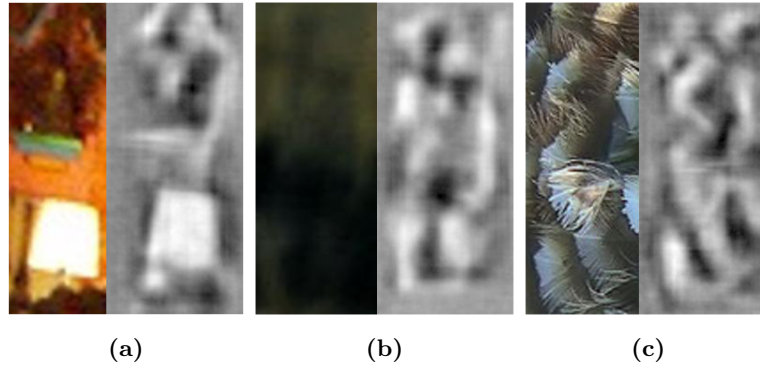


Figure 3.14: HOG HOGgles representation

3. A ROBUST FEATURE FOR OBJECT RECOGNITION

4

SVM

4.1 What is SVM?

In a wide variety of situations one may want to be able to assign a concrete object to one or more categories or classes, based on its characteristics. For example, given the characteristics of a concrete document page we may want to be able to determine, from a set of page types, which type of page it is. In computer science this is called a classification problem. SVM or *Support Vector Machine* are supervised learning models and algorithms that are able to analyze data and recognize patterns to automatically build a set of rules to classify similar data that they have never seen before.

By supervised learning we refer to the task of inferring the above mentioned set of rules from already labeled data. Other kind of algorithms are able to group or classify different object into clusters by similarity without knowing a priori the classes they belong to. In the case of SVM we need to provide labeled data for all the classes in which we wish to have our data classified.

4.2 How it works?

Let's explain the simpler case in which we only have two classes; generally called a binary classification.

Then, given a set of L labeled points, $\{x_i, y_i\}$ with $x_i \in \mathbb{R}^d$ being a vector of features or characteristics of the i -th object and $y_i \in \{-1, +1\}$ the class label, we want to build a rule to determine, given a new x , one of the two possible classes. We will also assume our data is linearly separable, this is to say that we can draw a line that splits all points

4. SVM

belonging to one class from the points belonging to the other class when $d = 2$, and a hyperplane when $d > 2$. Furthermore we want this separating hyperplane to maximize the margins or distances to the closest points of each class.

The above mentioned hyperplane can be described by $w \cdot x + b = 0$ where w is normal to the hyperplane and $\frac{b}{\|w\|}$ being the perpendicular distance from the hyperplane to the origin.

Then solving the classification can be written like:

$$x_i \cdot w + b \geq +1 \text{ for } y_i = +1 \quad (4.1)$$

$$x_i \cdot w + b \leq -1 \text{ for } y_i = -1 \quad (4.2)$$

This two equations can be described by only one equation in the following form:

$$y_i (x_i \cdot w + b) - 1 \geq 0 \quad \forall_i \quad (4.3)$$

If, as we said, the sets of points are linearly separable, then we can draw a line or hyperplane going through the points that lie closest to the separating hyperplane, this is, the support vectors. The main aim of SVM is then to maximize the distance between the new hyperplanes; let's call them H_1 and H_2 .

Figure 4.1 illustrates the above explained theory.

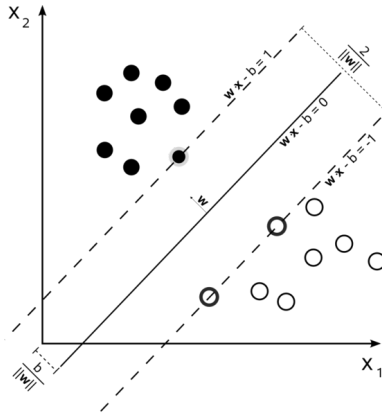


Figure 4.1: SVM maximum separating hyperplane margin -

Geometrically we can see that the distance from H_1 to the hyperplane is equal to $\frac{1}{\|w\|}$, and similarly to H_2 , so the distance from H_1 to H_2 is equal to $\frac{2}{\|w\|}$. As we explained the main goal is to maximize this distance, then we have that solving the problem reduces

to minimize $\|w\|$. Minimizing $\|w\|$ is equivalent to minimize $\frac{1}{2} \cdot \|w\|^2$ but using the later term makes possible to perform Quadratic Programming optimization (*QP*).

Therefore we have:

$$\begin{aligned} \min_{(w,b)} \quad & \frac{1}{2} \cdot \|w\|^2 \\ \text{s.t.} \quad & y_i(x_i \cdot w + b) - 1 \geq 0 \quad \forall_i \end{aligned} \tag{4.4}$$

The development of the *QP* problem is out of the scope of this thesis, but the complete development can be studied in detail in (18, 19, 20).

Regardless of the details of the *QP* the intuition behind SVM can be easily explained with figure 4.2, where we have three different planes. H_3 does not even separate completely the two set of points. H_1 does separate the points, but does not satisfy the maximum margin restriction, finally H_2 separates both sets and maximizes the distance from the hyperplane to the support vectors.

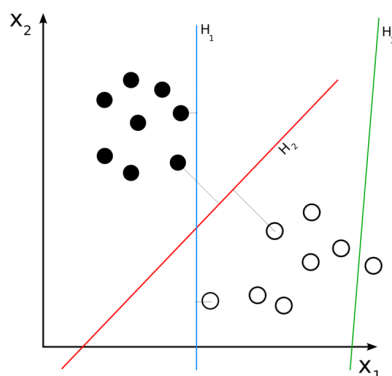


Figure 4.2: Different hyperplanes in 2D - This figure shows three different separations but only H_2 gives the maximum separation margin

4.2.1 SVM kernels

The reader will have noticed that, until now, all the possible separation boundaries that SVM is able to infer turn out to be lines in the \mathbb{R}^2 case or hyperplanes in \mathbb{R}^n . Sometimes more complex boundaries are needed in order to classify not linearly separable data. Figure 4.3 shows two examples of 2D representations of non linearly separable data.

4. SVM

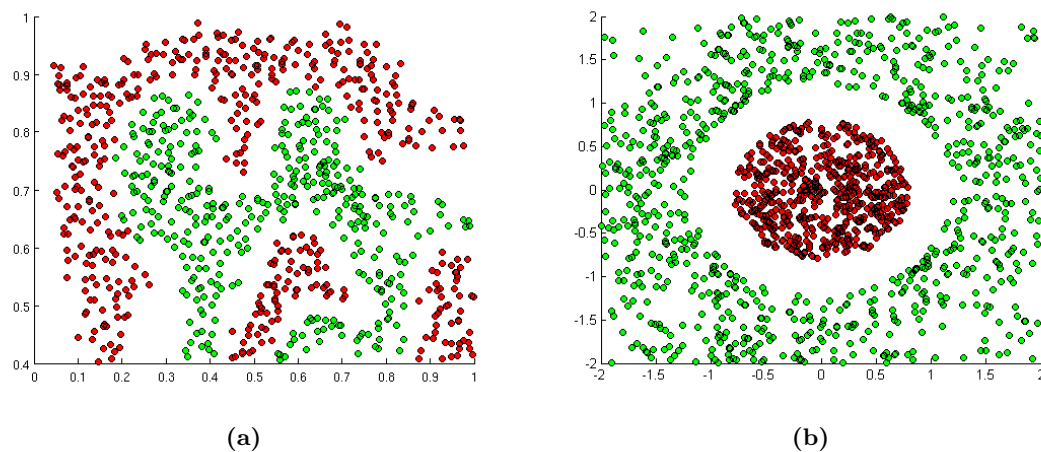


Figure 4.3: Graphic examples of 2D non linear separable data.

When dealing with this kind of data a more complex boundary is a clear need. The way in which SVM deals with this is through *kernel tricks*. Often non linearly separable features become linearly separable when mapped to a high dimensional feature space. This is an interesting property as allows us to use the same classifying method explained in §4.2 without an extra effort, the only requirement is to transform in some way our data.

Let give an example to illustrate this idea.

Figure 4.4 (a) shows the original data presented in figure 4.3, at its right we show the results of mapping all the points to a three dimensional space through:

$$\begin{aligned}\phi : \mathbb{R}^2 &\rightarrow \mathbb{R}^3 \\ (x_1, x_2) &\mapsto (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)\end{aligned}$$

Clearly after mapping a 3D feature space our data becomes separable by a hyperplane.

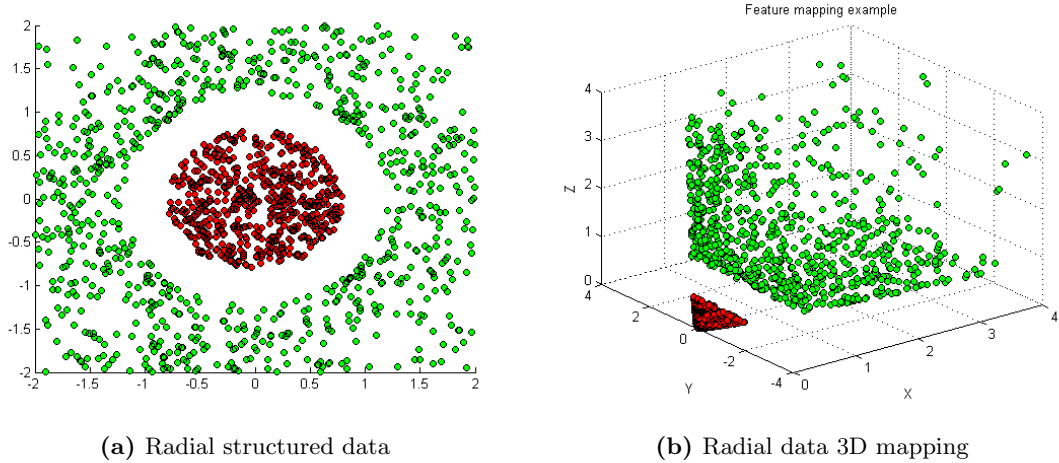


Figure 4.4: Feature mapping to a higher dimensional space

Although many kernels exist (*Fisher, graph, polynomial, string kernels etc*) we explain the Gaussian kernel or Radial Basis kernel (*RBF*) as this is the one used later on this thesis. The expression defining the RBF kernel follows:

$$K(x, x') = \phi(x)^T \phi(x') = \exp\left(-\frac{\|x - x'\|_2^2}{2\sigma^2}\right) \quad (4.5)$$

Figure 4.5: Gaussian kernel formula.

Where σ is a free parameter, x and x' are feature vectors in some input space and $\|x - x'\|_2^2$ is the squared Euclidean distance.

Sometimes can also be found written like:

$$K(x, x') = \exp(-\gamma \|x - x'\|_2^2)$$

$$\text{with } \gamma = \frac{1}{2\sigma}$$

Even though the kernel feature expansion has infinite number of dimensions ¹ we don't ever need to compute the feature mappings $\phi(x)$, actually we work with their kernels.

¹Assume that Gaussian RBF kernel $k(x, y)$ is defined on domain $X \times X$ where X contains an infinite number of vectors. One can prove that for any set of distinct vectors $x_1, \dots, x_m \in X$, the matrix $(k(x_i, x_j))_{m \times m}$ is not singular, which means that vectors $\phi(x_1), \dots, \phi(x_m)$ are linearly independent. Thus, a feature space H for the kernel k cannot have a finite number of dimensions.

4. SVM

As this kernel ranges from 0 (in the limit) to 1 (when $x = x'$), it can be seen as a measure of similarity.

The way in which the new similarity measure helps us follows:

Let's suppose we have m training instances. Then for every instance of our training data set, we define a set of *landmarks* in such a way that every *landmark* position corresponds exactly to the points of the training set.

Formally given $(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)$ training instances, we choose the mentioned *landmarks* as $l_1 = x_1, l_2 = x_2, \dots, l_m = x_m$.

Now given an example x , instead of having x as the feature vector, we map the vector to:

$$x \mapsto x_{mapped} = \begin{bmatrix} k(x, l_1) \\ k(x, l_2) \\ \vdots \\ k(x, l_m) \end{bmatrix}$$

Then the same optimization problem can be formulated but instead of x using the new feature vector x_{mapped} . In conclusion classifying a point boils down to predict one when:

$$x_{mapped} \cdot w + b - 1 \geq 0 \tag{4.6}$$

And 0 otherwise.

4.2.2 Training a SVM

While the above section explains the main theory behind SVM we have not explained yet the effect of some details and parameters involving the learning itself.

The effectiveness of SVM, assuming we have good data and features, depends on the kernel selection, the kernel parameter selection and on the soft margin parameter C . Between the kernel parameters, in the RBF case, we find the γ value, which we will explain further on. We have yet explained nothing about the soft margin parameter C until now. The main idea behind C is to allow some amount of miss classified instances when no possible clean separation can be done by any hyperplane, but still achieve the cleanest possible separation. One way of understanding C is to see it like a parameter to measure the effort that SVM will do until finding a separation boundary. Formally it is a weighting parameter to control how much penalizes every miss classified instance.

We can write the objective of SVM like:

$$\begin{aligned} \min_{(w,b,\xi)} \quad & \frac{1}{2} \cdot \|w\|^2 + C \sum_{i=1}^m \xi_i \\ \text{s.t.} \quad & y_i(x_i \cdot w + b) - 1 \geq -\xi_i \quad \forall_i \quad \xi_i \geq 0 \end{aligned} \quad (4.7)$$

Figure 4.6 shows the effect of the C parameter, if we strongly penalize a miss classified instance, SVM will try to perfectly fit every instance, giving as separating hyperplane H_1 what will lead to have a poorer classification boundary. If instead, we relax the cost value, we obtain a more reasonable hyperplane, H_2 .

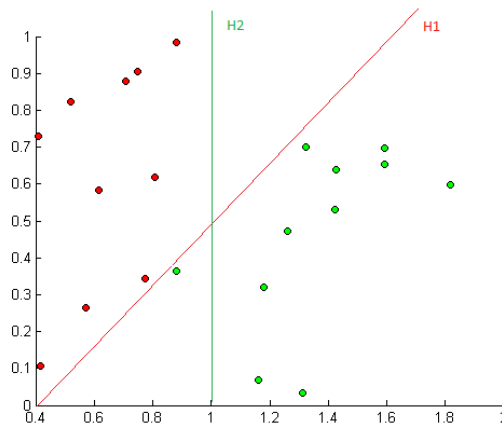


Figure 4.6: Cost parameter effect - H_1 corresponds to a high C value, while H_2 correspond to a low C value

With respect to γ , the relationship with σ is $\gamma = \frac{1}{2\sigma^2}$. Therefore when σ is large (γ small) we have that the similarity function varies smoothly, conversely σ small (γ large) implies a more pronounced variation in the similarity function. This can be seen by plotting the shape that takes the Gaussian function when varying σ and taking the similarity value as the height of a concrete point (figure 4.7). Furthermore figure 4.8 illustrates the effect on the classification boundary.

4. SVM

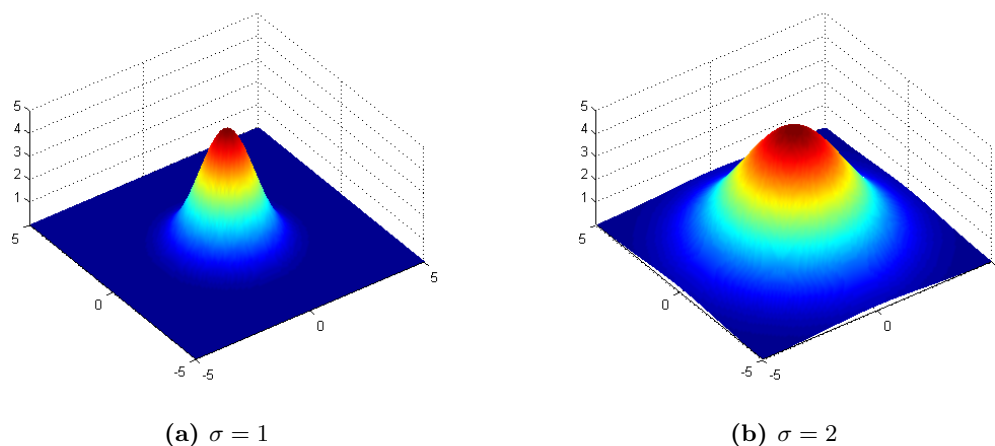


Figure 4.7: Gaussian curves as similarity functions

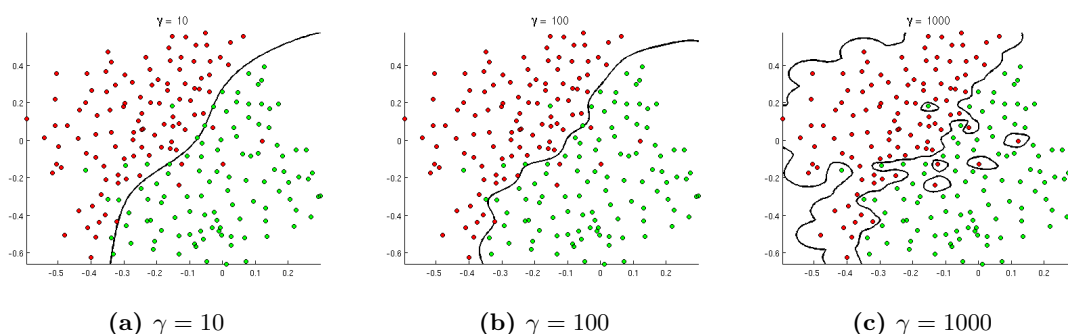


Figure 4.8: γ effect on the classification boundary

4.2.2.1 Selecting the SVM parameters

Now we have reviewed the effects of the parameter selection and the impact of these in the SVM boundaries, we can talk about how to choose the proper values to each parameter and which analysis we can perform in order to detect some problems like over-fitting or under-fitting.

Many times classification problems involve features with more than two dimensions, therefore training a SVM with some pair of values of *gamma* and *C* and plotting the boundary is not possible, moreover does not seem a very intelligent procedure. The most accepted way to find the optimum parameters is to perform a cross validation over a range of possible values. Mainly this is a technique to check how well will a

model generalize on unseen data. Given a set Γ of possible γ values, a set κ of cost values and an integer k , the cross validation method, for every parameter configuration in $\Gamma \times \kappa$ builds a model with a random subset of size $\frac{k-1}{k}$ and with the remaining $\frac{1}{k}$ validates the model. This process is repeated k times (*folds*) in such a way that every fold is used once as validation set. From each repetition an accuracy measure is obtained, the average accuracy is then the accuracy of the model for that concrete parameter selection. Even though this is the most popularized technique for performing a parameter search, some problems may arise, the most common are:

Over-fitting.

We say that the model has over-fitted when describes random errors or outliers rather than the underlying relationship between data. The possibility of over-fitting exists because the criterion used for training the model is not the same as the criterion used to judge the efficacy of a model. When training we try to minimizing the error in classifying a set of instances, this is, to fit some concrete set of points, but what really makes good a model is its capability of generalization and therefore how well classifies unseen data. Not only a bad parameter selection is likely to lead in an overfit, also having a little number of training instances is likely to over-fit, specially if using a complex boundary model.

Figures 4.6 and 4.8 (c) clearly show examples of over-fitting. The first one due to a large cost and the second because of a too large γ . Owing to this behavior it is usual to find that, when training RBF models, the optimum parameter configurations present some kind of diagonal through the parameter space, as lower γ values tend to balance the effect of large C values and vice versa.

Diagnosing over-fitting is not always easy. An evidence of over-fitting, due to a bad parameter selection, might be having a low error when classifying the training set but a high error when classifying the cross validation or test set. Plotting error curves in function of the C value for both training and cross validating sets may give some clue. Figure 4.9 shows an example of the mentioned learning curve.

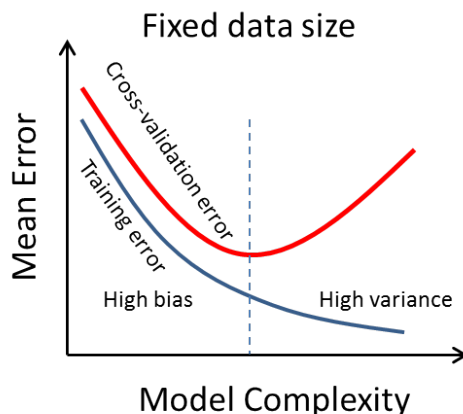


Figure 4.9: Learning curve showing the evolution of the error in function of the model complexity - High variance = over-fit, high bias = under-fit

Another reason for over-fitting could be an insufficient number of training instances. Plotting error curves in function of the training instances for both training and cross validation or test set could help. In this case we expect to see the training error increasing as the number of training instances grows and the cross validation error decreasing, as the model is not possible to perfectly fit the total number of instances. It is also common to see a large gap between both type of errors. Figure 4.10 show an example of the expected curve.

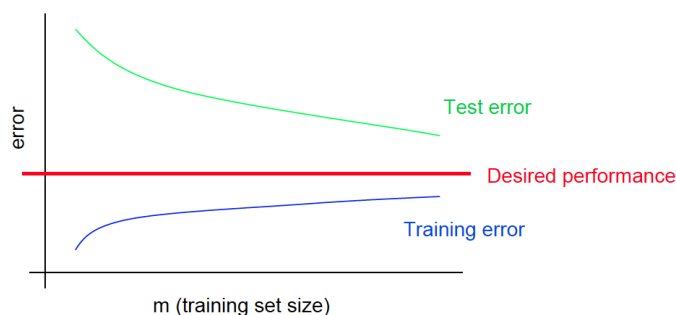


Figure 4.10: Typical learning curve for high variance - Large gap between errors and error convergence tendency suggest over-fitting

In conclusion to solve this problem we should opt to increase the number of training instances, what is not always possible and usually implies a hard labeling work, or to relax the model complexity, either choosing another parameter configuration or using

another kernel.

Under-fitting.

On the other hand we say a model has under-fitted when it is not capable of describing the relationships between the given data. Under-fitting may occur when trying to fit some complex data with an excessive simple model. For instance trying to classify with a linear kernel the data shown in 4.3 would lead in a clear case of under-fitting, as a line is not able to split both classes. Contrary to what happens in the over-fitting case, when the model turns out to be very simple, both training and testing errors are high. The same analysis can be done to determine if we have an under-fitting problem. Plotting error curves in function of the training set size should help. Figure 4.11 show the typical shape these curve should show when suffering from under-fitting. In this case usually a little gap separates both errors.

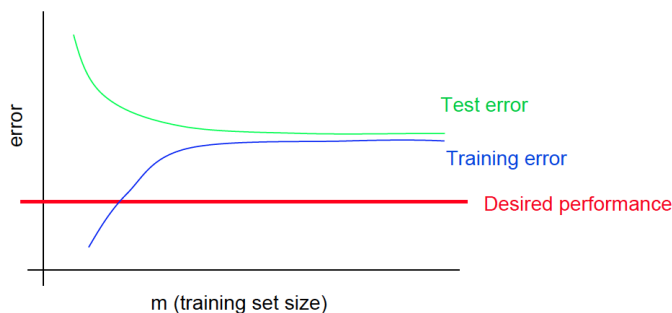


Figure 4.11: Typical learning curve for high bias - Little gap between errors and high errors suggest under-fitting

In this case, increasing the training instances, by itself, does not tend to help. When dealing with this problem we should increase the model complexity.

4.2.3 Testing a SVM

Some considerations must be taken into account when trying to determine the effectiveness of a classification model. The first, and most important, is to select as testing instances data not used in the training process. As we want to test the real performance of the model we need to know how it does when dealing with unseen data, this should give us an intuition of how did the model generalized while training.

4. SVM

The second consideration, has to do with the relation between the amount of instances from each class. Let's think we test with a very skewed data set where 99% of our data belong to the positive class and the remaining 1% to the negative one. Now we perform our test with a model, that independently of the input, always outputs a positive prediction. In this scenario, counting the right guesses will give a 99% of accuracy, however it is obvious this model is not a good choice. Therefore is preferable to use a similar amount of instances of each class. However some measures apart from a simple count can be done to circumvent this drawback if for some reason we specially need to use skewed data sets. These measures and their explanations can be seen in §5.2.

5

Pedestrian detection

5.1 Training the detector

A complete explanation follows about how the SVM training has been done, not only of the final detector, also of all the intermediate states, models and detectors that have led the way until achieving the definitive one.

In general every detector trained is divided in two or more steps of retraining. First a pre-model is trained, then several hard example searches follow using the previous trained model to achieve this purpose, this is, using the just trained model a exhaustively search along the negative images is done with the purpose of finding windows that the model mistakes in its classification. Each model in turn is trained searching through a cross-validation the optimum parameter sets for that kernel within a range of possible values for each parameter. Once the best parameter within the given parameter space is found that model is saved and becomes the model for the hard example search. This process is repeated until no further improvement is reached or the performance increases in a very low rate.

When talking about exhaustive search we refer to a search in a dense space-scale pyramid following a sliding window paradigm. See figure 5.1

The starting scale level is 1, so the first image is the original one, then we keep adding one more level in the pyramid until the size of the scaled image is greater than 64 for the horizontal axis and 128 for the vertical axis. The scale ratio between consecutive levels is 1.2. So concretely we will generate more scaled images until $\left\lfloor \frac{\text{ImageWidth}}{\text{Scale}} \right\rfloor > 64$ and $\left\lfloor \frac{\text{ImageHeight}}{\text{Scale}} \right\rfloor > 128$.

5. PEDESTRIAN DETECTION

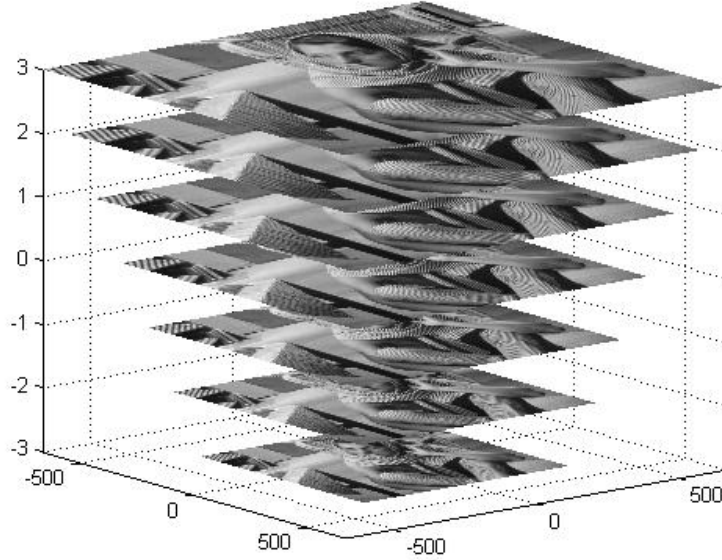


Figure 5.1: Scale-space pyramid illustration - Each level represents a scaled version of the original image

As regards to the sliding window configuration, the window stride (sampling distance between two consecutive windows) at any scale is 8 pixels. If after fitting all windows at a scale level some margin remains at borders, we divide the margin by 2, take its floor and shift the whole window grid.

For example, if image size at current level is (75,130) and the margin (with stride of 8 and window size of (64,128) left is (3,2), then we shift all windows by $\left\lfloor \frac{\text{MarginX}}{2} \right\rfloor$, $\left\lfloor \frac{\text{MarginY}}{2} \right\rfloor$.

New image width and height are calculated using the formulas:

$$\text{NewWidth} = \left\lfloor \frac{\text{OrigWidth}}{\text{Scale}} \right\rfloor \text{ and } \text{NewHeight} = \left\lfloor \frac{\text{OrigHeight}}{\text{Scale}} \right\rfloor.$$

Here scale = 1 implies the original image size.

5.1.1 Data sets

As pedestrian in particular, and any object detection in general, is becoming an import goal in the machine learning field, several image data sets can be found all over internet, many of them belonging to computer vision institutes and research departments all around the world. For this thesis two well known pedestrian data sets has been used,

the first one from the *Massachusetts Institute of Technology* (MIT) and the second one from the *French National Institute for Research in Computer Science and Control* (INRIA).

This are the specifications of each data set:

- MIT Data Set: (21)
 - 64 x 128 (x3) PPM format images
 - 924 files (positive right standing pedestrians)
 - 10 Megabytes compressed : 22 Megabytes uncompressed
 - pedestrian poses limited to rear or front views (people height from shoulders to the feet is approximately 80 px)
 - images obtained from color video sequences taken in different seasons with different video cameras

- INRIA Data Set: (22) Divided in two formats. (a) original images with corresponding annotation files, and (b) positive images in normalized 64x128 pixel format (as used in (8)) with original negative images.
 - 70 x 134 normalized and centered positive test images (left and right reflections)
 - 96 x 160 normalized and centered positive train images (left and right reflections)
 - 1218 original negative training images
 - 614 original positive training images
 - 453 original negative testing images
 - 288 original positive testing images
 - 970 Megabytes compressed : 1150 Megabytes uncompressed
 - Only upright persons with height larger than 100 px are marked in each image
 - images obtained from different sources

5. PEDESTRIAN DETECTION

5.1.2 SVM models

Several SVM models has been trained to find such a configuration that performs in the best possible way.

5.1.2.1 MIT models

For the MIT data set linear kernel and color images has been used because perfectly separation was achieved. Results can be found in the test section. As the MIT data set only provides positive images an additional group of 2000 random negative images were sampled from images were no pedestrian could be found. This images joined with a group of 654 positive images (the positive images given as train set by MIT data set) were fed into a linear SVM to train a pre-model.

To find a reasonable good pre-model various models were trained with different parameter configuration performing a 5-fold cross-validation over the training set to find a measure of the performance depending on the parameter. As the model was a linear kernel model the only parameter to adjust is the cost parameter (C) as explained in §4.2.2. This parameter searches were made over a power of two range starting from 2^{-2} and up to 2^7 , finding out that increasing C over a value of 2^3 makes the model overfit and decreases its performance as can be seen in figure 5.2.

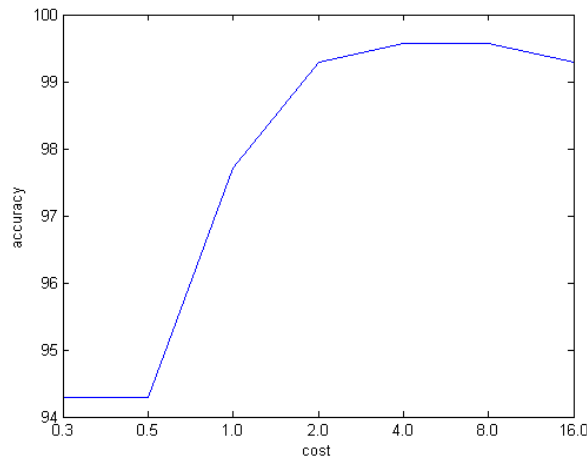


Figure 5.2: MIT pre linear model cross-validation - y-axis represents performance and x-axis cost value.

Although this model is just a preliminary model reaches a cross-validation accuracy of

99.85% and even though the cross-validation was performed over the train set it gives a good intuition about how will the model perform on unseen data.

The final model was acquired by searching exhaustively, as explained at the beginning of this chapter, the original 2000 negative images from where the negative training set was sampled to find hard examples. Then the initial training set plus the hard examples found by the pre-model were fed into the linear SVM to obtain the final model repeating the above explained process. This definitive MIT model performed slightly better than the preliminary one, this slight difference can be seen in the next cross validation.

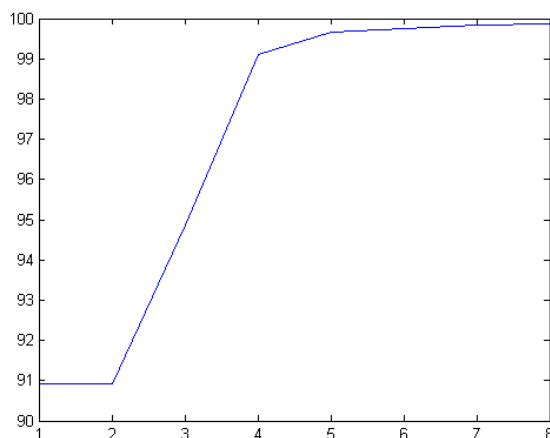


Figure 5.3: MIT round 1 linear model cross-validation - y-axis represents performance and x-axis cost value.

5.1.2.2 INRIA models

Linear models. For the more challenging INRIA data set we can find two groups of models depending on the SVM kernel used.

For the purpose of training SVM models from the INRIA data set a bunch of 12180 negative windows were sampled randomly from the original negative images.

As the training process remains the same as long as the SVM kernel type is the same, the training process will only be explained once for each kernel type even though all the results will be shown later on this chapter.

The process for all the linear kernels of the INRIA data sets are identical to the process explained for the MIT data set so it can be considered as explained but considering a

5. PEDESTRIAN DETECTION

little difference in the cross validation process. While in the MIT models a 5-fold cross validation was performed, in the INRIA models a 3-fold cross validation instead was done, this was like this because of the difference in the number of images used to create the models.

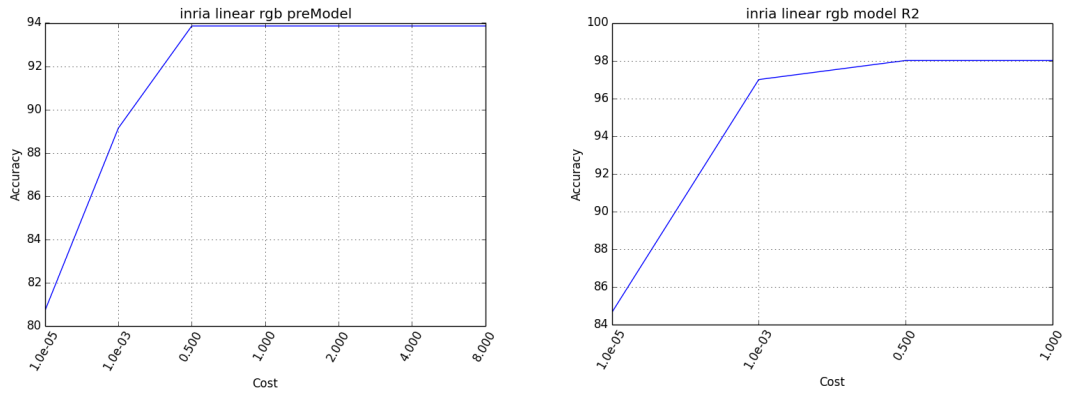
Due to the great amount of images involved in the training process and because of the nature of the training itself a big amount of time and memory is needed to perform a more high order fold cross validation. This fact is specially notorious in the first retrain round when the hard examples found by the pre model are used. With this in mind and at the expense of a slightly poorer error estimation in the training process the number of folds was reduced from 5 to 3.

As the training method followed by the authors was not very clear explained with respect of the number of negative instances used to train the pre-models, two ways of proceeding were tried.

In the first approach a subset of the whole negative training images was chosen randomly to train a pre-model, once this model was ready the hard examples were joined with the negative training images and served like the training set for the different rounds. Generally a similar number of negative and positive instances were used.

It is noteworthy that, in contrast to the MIT case, the first linear model of the INRIA data set did not perform as good as the MIT case did, so the re-train step becomes more important. Anyway only one retrain round is worth it, further rounds do not improve significantly the performance of the detector. The cross-validation curves of the RGB models can be seen in figure 5.4.

5.1 Training the detector



(a) INRIA pre linear model cross-validation

(b) INRIA round 1 linear model cross-validation

Figure 5.4: Cross validation accuracy tendency round comparison

y-axis represents performance and x-axis cost value.

The second approach consists in the same process but using as initial set the whole negative training set provided by the INRIA team. In this way a more wide range of cost values was needed to find the optimum parameter configuration, leading in the following parameter search and curve. 5.5

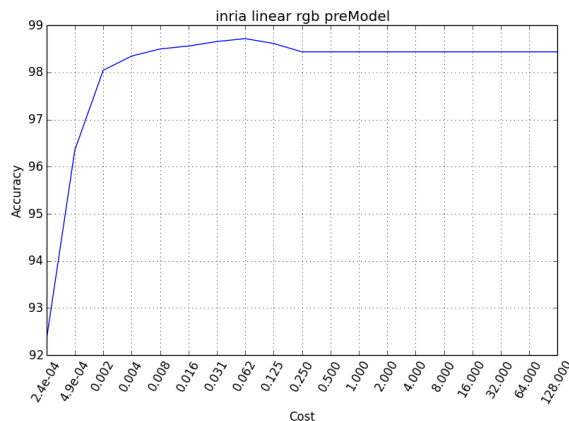


Figure 5.5: INRIA pre linear model cross-validation using all the negative training instances - y-axis represents performance and x-axis cost value.

5. PEDESTRIAN DETECTION

As can be seen the larger data set needs a softer cost condition. This can be understood because finding a more adjusted line, fitting better the training set, can lead in a poorer generalizing conclusion and when unseen data is tested the model fails to correctly separate between classes.

In this case retraining the above pre-model did not make any improvement, anyhow the above cross validation results show very similar results for both approaches, even slightly better results can be seen in the former one. 5.6 shows the cross validation curve.

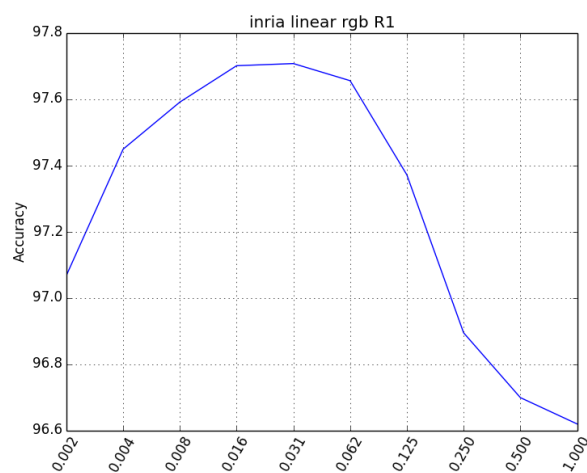


Figure 5.6: INRIA linear model round 1 cross-validation - y-axis represents performance and x-axis cost value.

Eventually for the linear kernel a gray scale model was trained. For the gray scale model same curves were achieved, showing the same tendency through the parameter space but with slightly lower cross validation accuracy. This decrease in the accuracy was constant over the whole curve and around 0.25% less. This was made just to be sure about the little impact that color images versus gray scale images have. In the tests section of this work more accurate experiments about using color versus gray scale images and models will be explained.

RBF models. Also RBF models were trained following the same methodology, this is, training a pre-model and the further exhaustive search through the original training images.

Despite of the fact that the process is always the same, in the RBF case a more intelligent way of feeding the SVM is needed. If at the very first attempt of training the pre-model we feed the entire negative image set, as the image set presents a important skew in the number of instances available between the positive and negative classes, along a large cost range of values, the cross validation accuracy results to be always the same, even for relatively high values of C and low values of γ , giving a value of 83.44%. A fast observation reveals that doing like this results in the SVM process considering only one class, in this case the negative one. This can be easily seen by dividing the negative training cases by the total number of instances, this is: $\frac{12180}{12180+2416} = 0.8344$. Given this behavior, we could opt to continue increasing the cost values or to train a pre-model providing SVM with a similar amount of negative and positives instances so neither of the classes hides the other one at the training phase. We consider both options even though a large time and memory requirements are needed when dealing with RBF cross validations and a huge amount of data. This leads again in two approaches and their corresponding sets of models.

As has been explained a main difference between the two families of models is the parameter space, for the *RBF* models a surface must be explored to find a reasonable good parameter set. This exploration for all the models trained consisted in a general and wide grid given, for both γ and cost parameters, a range going from 2^{-2} up to 2^6 in step of powers of 2 is used.

After this general cross validation accuracy grid is done, if needed, a more fine-scaled parameter search is made, looking this time, only over the possible promising areas. This areas could be, for instance, the gaps between two values of cost or γ of the previous cross validation or any possible value outside the first parameter range.

Regarding the first approach, is to say, using a sub set of the training images where negative and positives instance are similar in number, three models where trained, two pre-models and the final round-1 model. The two pre-models stand for different parameter searches over areas far away each other to figure out which was the most promising one. Figure 5.7 show the wide parameter searches made for the pre-model

5. PEDESTRIAN DETECTION

and the round-1 model. Both searches show high accuracy values, but about a 3% of improvement is achieved by the re-training step.

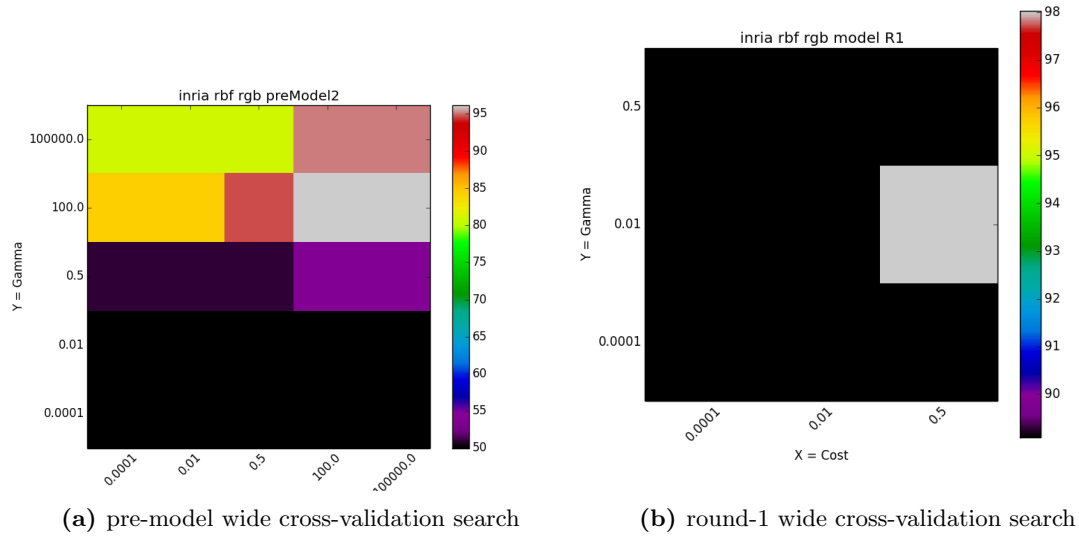


Figure 5.7: RBF parameter cross validation search (1st approach)

For the second approach, we performed several parameter searches until finding the most promising configuration, the cross-validation search grid plots can be seen in 5.8 and 5.9.

5.8 shows the results of the first and general search, used to figure out, in a first approach, how it looks the parameter surface.

As this surface throws no good configurations, a wider search is done, this time trying some extreme values for the parameter, hoping this would shows any tendency and therefor get some clue from where to search. Once the tendency could be spotted out a finer search was done. Concretely for this model we can see how large cost values but low gamma are needed to obtain a good performance. This can be appreciated in 5.9

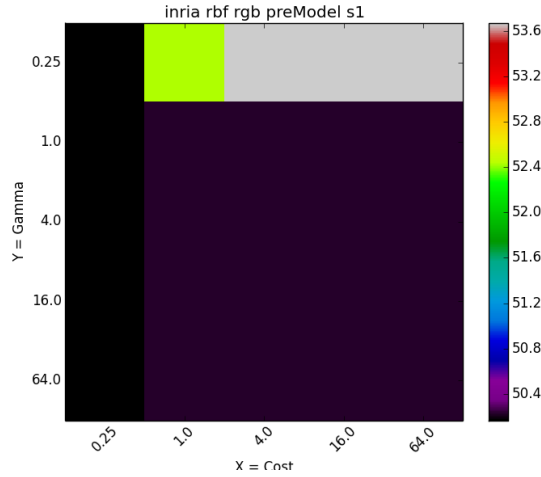


Figure 5.8: First RBF parameter search - y-axis represents gamma and x-axis cost value.

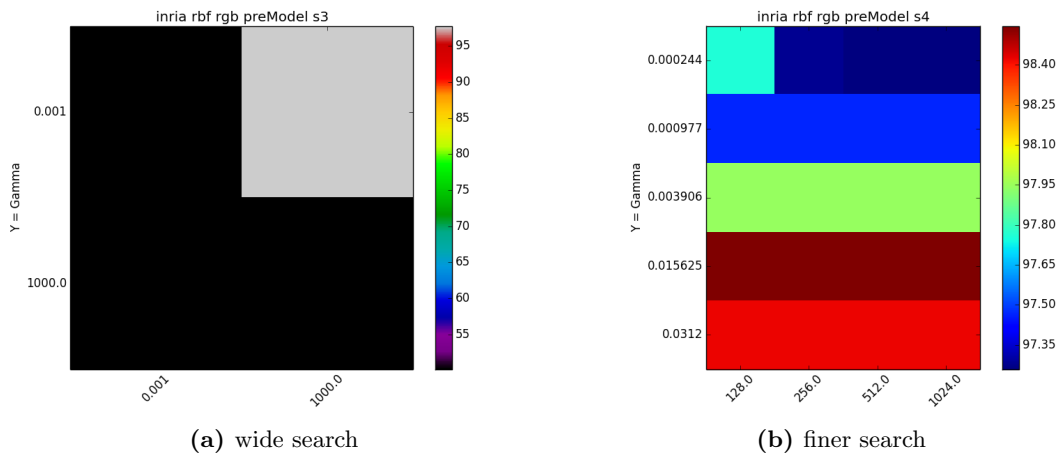


Figure 5.9: RBF parameter cross validation search (2nd approach)

Then as usual, the original negative images are exhaustively searched to find hard examples. This examples then become part of the training set and the entire process is repeated. Anyway this did not represent an improvement, slightly worst accuracies were achieved around the expected areas in the parameter space where good results should be found.

It can be seen that a higher accuracy is reached, compared to the accuracy achieved by

5. PEDESTRIAN DETECTION

the first approach with either linear and Gaussian kernels. But it is worth mentioning that is at a expense of a really higher computational time.

5.2 Testing the detector

Even though the cross validations performed during the training stages give a quite accurate measure of the performance of a model, several tests has been performed to each model to really know how it will respond with unseen data.

Between the main reasons to develop good test methods and measures the next one could be remarked, while for cross validating not the entire training data is used to find accuracies, as the training set is divided in fractions as explained in §4.2.2., this will make some differences between the numbers obtained in the training stages and the definitive model.

Another important aspect to note is that in the case of this work, the accuracies computed at cross validations does not take into account possible differences in the importance of misses, so a false positive is as bad as a false negative. For the concrete case of pedestrian detection a mistake in the classification of a window containing no pedestrians, resulting as a positive case, may be less important that the opposite case. As this work is centered in the detection itself and not in any concrete application of pedestrian detection, all mistakes in the classification are counted as equal. Therefore its possible to find models with the same accuracy but differing in the type of errors committed. Furthermore, when training with image set that greatly differ in the instance number between classes, high accuracies can be reached while cross validating but only because a class can represent a good percentage of the whole training instances.

Knowing the foregoing considerations this performance measures has been done to each model:

- Precision, Recall and F-score measures
- Receiver Operating Characteristic (*ROC curve*)
- Detection Error Trade-off curve (*DET curve*)
- SVM scores distribution analysis

An explanation about what is each measure and how it is obtained follows.

Precision, recall and F-score measures. In pattern recognition and information retrieval with binary classifications two different measures can be done. We understand by precision the fraction of retrieved instances that are relevant while recall can be understood as the fraction of relevant instances that are retrieved. In a classification task we would define:

- t_p : true positives, the number of correctly labeled as belonging to the positive class
- t_n : true negatives, the number of correctly labeled as belonging to the negative class
- ft_p : false positives, the number of wrong labeled as belonging to the positive class
- fn : false negatives, the number of wrong labeled as belonging to the negative class

Then:

$$\text{precision} = \frac{t_p}{t_p + ft_p} \quad (5.1a)$$

$$\text{recall} = \frac{t_p}{t_p + fn} \quad (5.1b)$$

Figure 5.10: precision and recall equations

A value of one in the precision measure means that actually every instance classified as positive was indeed positive but tells nothing about the items that did not were classified as positives. On the other hand a value of one in the recall measure means that all the positive instances were classified as positives but says nothing about the negatives instances that were also label as positives.

As has been pointed out by the definition of the above measures a kind of average is needed to somehow easily compare results from different models. A measure that combines precision and recall is the harmonic mean of both values, traditionally called F-score.

$$F_{score} = \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad (5.2)$$

Figure 5.11: F-score equation

This is also known as F1 score as recall and precision are evenly weighted.

5. PEDESTRIAN DETECTION

ROC curves. When there is a tradeoff of error types, a single performance number is not the best solution to represent the capabilities of a system. Such a system has many operating points, and is best represented by a performance curve. A receiver operating characteristic, or simply ROC curve, is a graphical plot which illustrates the performance of a binary classifier system as its discrimination threshold is varied. It is a plot of the true positive rate against the false positive rate, then, the tradeoff between sensitivity and specificity (any increase in sensitivity will be accompanied by a decrease in specificity).

Some observations should help to interpret this plot:

- The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test.
- The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.
- The slope of the tangent line at a point gives the likelihood ratio for that value of the test.
- The area under the curve is a measure of accuracy. An area of 1 represents a perfect test; an area of .5 represents a worthless test.

Anyhow when a good performance is achieved most of the plotting area is underutilized as the curve tends to get closer to the left and upper margins, giving fewer details about the tradeoff between errors. Therefore we also use other type of plot to be able to compare in detail different models.

DET curves. In order to solve this issue with ROC curves we also use DET curves. In the DET curve we plot error rates on both axes, giving uniform treatment to both types of error, and use a scale for both axes which spreads out the plot and better distinguishes different well performing systems, this usually produces plots that are close to linear.

This plot assumes a normal likelihood distribution for both positives and negatives instances and scales the axis according to this assumption. This scaling and linearity allows to have a clearer observation of the system behavior.

Besides this advantage, some special points can be easily viewed in DET curve. A weighted average of the missed detection and false positives rates may be used as a reference of the overall performance, in this work all the plotted curves indicate, by a little circle, the point where this average becomes minimum. As we don't have any special requirements in minimizing any of the two types of errors an evenly weighted average is performed to find the point where this measure becomes minimum, anyhow this could be adjusted to meet any special requirement that a concrete application may need.

SVM scores distribution. In the general case SVM defines one or more hyperplanes which behave as boundaries between classes, therefore the confidence of belonging to a concrete class can be seen as the distance from a boundary. This is what we call classification score. Just for having a clue on how well the boundaries were placed, we plot a histogram counting how many positive and negative instances were at a similar distance from the boundary using different colors between classes. In this plots we expect to find little overlapping between different color bins as SVM tries to maximize the margins from the boundary to each class. We could also use the probability estimates but it turns out to be a expensive computation and is know to have some numerical issues. Even though a score may not be a very informative measure itself, we are not interested in the value itself but in the distribution of all the scores.

Threshold curves. *SVM* implicitly uses a 0.5 threshold to determine whether a concrete instance belongs to one class or the other. Depending on how the prediction probabilities are distributed and the actual ground truth values, maybe other thresholds may prove to be more appropriate. To find the optimum threshold for a desired trade-off between the two possible kind of errors (false positives and false negatives) we plot precision, recall and F-score measures in function of the classification threshold.

5.3 The final detector

Once the best possible model has been selected a complete detector can be made. The detector consists then in a SVM classifier using the best model achieved in the previous training stages plus a sliding window system.

5. PEDESTRIAN DETECTION

Given an image, we re-scale the image, then a dense scale-space pyramid is made and a 64×128 detection window scans all the pyramid levels sliding this window as explained in the process of the exhaustive search performed during the training for finding hard examples.

This re-scaling of the input image is done for two main reasons. The first reason has to do with the way on how the model was trained. As all the models were trained with 64×128 images where pedestrians presented an average height of 100 pixels, the testing stage must also be compliant with this. This reduces the overall processing time by not checking non useful windows.

After this reduction is made, the exhaustive search begins. As one pedestrian can cause several detections from nearby windows a non maximal suppression is applied so only the most probable detection window is shown. Anyway the detector is capable of drawing all the detection window if desired as well as showing each window in separate figures for a more detailed examination.

Just for the purpose of understanding and making visible how the sliding window method works, we made another detector where the sliding windows is shown at every stage, showing a red rectangle where no pedestrian is found inside the window and green rectangle with the percentage of confidence when a pedestrian is found.

Regarding to the non-maximum-suppression algorithm, the main idea behind it, is to group all the detection windows fulfilling some proximity condition for comparing their classification probabilities and suppress all but the most probable detection within each cluster.

We tried two proximity condition to group detection windows. The first approach is to group them by a squared Euclidean distance measured in pixels. For every window we compute its distance as $d = d_x^2 + d_y^2$. Where d_x and d_y are the coordinate difference between the top left corners of each window. As the square root operation turns out to be computationally expensive and does not provide any improvement to the calculus we decided not to do it. Every window at a shorter distance than the square of the shortest side of the detection windows size is considered near each other.

The other approach is similar but comparing the overlapping area between windows, every two windows with a overlapped area greater than half the total area is considered close and therefore considered as two windows detecting the same pedestrian.

5.3 The final detector



Figure 5.12: Multi detection versus non-max-suppression detection

Appears that the first approach gives better suppressions for the tested images. Anyway some more detailed testing should be considered as the proximity conditions could not be the optimum ones.

Some detection suppression examples are shown in figure 5.12.

5. PEDESTRIAN DETECTION

6

Implementation and Usage

In this section we give a brief review of the implementation and some details of each detector component, nevertheless a list and explanation of the libraries and packages used in the project can be found in §9.

Additionally to the description in this chapter a complete documentation in *HTML* format is provided to easily navigate through the whole set of functions, making possible to see the calls between them and the complete code.

The development of the entire project has been made using *MATLAB*, a widely known language and environment, designed for numerical calculus and fast development of applications. Between the main reasons for what *MATLAB* was chosen, we find:

1. Much faster development compared with traditional languages like C++ or Java
2. Large number of libraries and toolbox for computer vision, optimization etc
3. Efficient manipulation of matrices and vectors
4. Large community and support

Besides *MATLAB* provides an interesting development environment, as *MATLAB* runs on top the Java Virtual Machine it's performance becomes poor compared with a more low level language.

MATLAB structures all the code through *.m* files that can be scripts or functions. The main difference between them is that functions expect input parameters and in turn returns any number of output parameter while scripts don't use neither input nor output parameters and therefore work with the workspace variables, so no private

6. IMPLEMENTATION AND USAGE

variables are available when working with scripts.

(A detailed explanation about the workspace and MATLAB can be found in §9)

In addition to the *MATLAB* project code, some scripts are also available in order to make possible some fast and easy results acquisition. Mainly we provide functions for plotting cross-validation grids and curves and create *excel worksheets* from different logging data written by the training or testing functions of the project itself. Many of this scripts are written in *Python* but its use is generally straightforward and brief explanations are included with the code.

6.1 Implementation

Before explaining the functions and script composing the project is worth being familiar with some *MATLAB* variable types, libraries or concepts as: cell array (9.1.1), .mat files (9.1.1) and the libSVM library (9.2.1).

Is also noteworthy that the whole project is made in such a way that every configurable parameter of the studied method is capable of being changed through configuration files. The functions and scripts composing the project follow (in alphabetical order):

- **compute_cell_coordinates.m** - function for computing cell coordinates given the cell x and y size. Input: image to split in cells, x cell size and y cell size.
- **compute_gradient.m** - function for computing the gradient of the input image. If the input image is a RGB image, the gradient is computed for each channel and the returned magnitudes and angles correspond to the highest magnitude for each pixel. Input: Image from where to compute the gradient. Output: Angles and magnitudes of the gradient in each pixel.
- **compute_HOG.m** - function responsible of computing the descriptor. Calls the compute_gradient function and performs the histogram computation and block normalization. Input: Image to process, cell size in pixels, block size in cells and number of bins of the histogram. Output: HoG descriptor of the image.
- **convert2gray.m** - Script for saving a gray scale version of every image in a specified folder. Used to obtain a gray set from the RGB images in train and test sets from INRIA person dataset.

- **cross_validate.m** - function responsible of the cross validation of a SVM model. Once the whole parameter space is searched the best configuration found is returned. Also a cross validation log file, where the accuracy reached for each configuration is logged, is saved along with a cross validation grid or curve representing the evolution of the accuracy with respect to the parameter selection. Input: kernel type (RBF or linear as string), cost and gamma ranges as lists of doubles, the train matrix (as many rows as training instances and as many columns as the HoG dimension), a label column vector specifying the class of each instance and the model save path. Output: String representing the best parameter set in the libSVM format.
- **static_detector.m** - function that performs the pedestrian search over all the images found in a specified folder. Given a model asks for a folder from where to read the desired images. Then for each image searches for pedestrians. If desired applies non-maximum suppression to draw only the most probable matches over the image marked with a green bounding box. If non-maximal suppression is desired all positive matches can be drawn. Also every detection can be shown in a separate figure for a more accurate examination if wanted. Input: SVM model. Output: Empty
- **sliding_detector.m** - function similar to *static_detector* but draws the scanning window all along its sliding process over every image. Prompts for the image folder path and calls *draw_sliding_window* for every image found in the folder. Input: SVM model. Output: Empty
- **draw_sliding_window.m** - function responsible of drawing the sliding window. Called from *sliding_detector*. Responsible of computing the descriptor and classifying each window as long as the detection window slides through the image. Input: sub image defined by the detection window and the libSVM model desired for the classification. Output: Empty
- **get_feature_matrix.m** - function to compute the descriptor matrix for all the input images. Input: Paths to positive and negative images. All window parameters are read from the *window_params* file. Output: labels; a column vector with the ground truth class for each input instance, train_matrix; descriptor/feature

6. IMPLEMENTATION AND USAGE

matrix where the number of rows is equal to the number of input instances and the number of columns is equal to the feature vector dimension.

- **get_files.m** - function for retrieving all or a subset of the image paths positive and negative folders. Input: number of positive image paths desired, number of negative image paths desired (-1 in case of asking for all the images in the folder) and a cell array containing the paths to the folders containing positive and negative images respectively. In case no folder paths are given a windows prompts for both folders path. Output: Two lists containing the positives and negatives paths to the images.
- **get_negative_windows.m** - function for computing the descriptor matrix for all input images. Input: two lists containing positive and negative images paths respectively. Output: labels; a column matrix with 1 in positives images and -1 in negatives, train_matrix; the descriptor/feature matrix. where num. Rows = num. instances and num. columns = feature dimension.
- **get_params.m** - function responsible of reading different parameters from a *.mat* file. Input: parameter file path. Output: map or dictionary mapping all the parameter keys to its actual values.
- **get_pyramid_dimensions.m** - function for computing the dimensions of the scale-space pyramid given an input Image. The parameters defining the stride, scale factor and windows dimensions are read from a pyramid parameter file. By default this file is searched in the *.m* file root folder, if not found is searched in the *params* folder where the *.m* file is located. If both searches fail to find the parameters file then a window prompts for it. Accordingly to the parameters read the function computed how many detection windows will the pyramid have. Input: Image to process. All pyramid parameters are read from the *pyramid_params* file. Output: number of levels, total number of windows and windows per level.
- **get_pyramid_hogs.m** - function responsible of computing all the HoGs from the space-scaled pyramid given an input image and the pyramid configuration parameters. Input: input image, descriptor size, pyramid scale factor between levels and window stride. Output: HoGs for every window, all the pyramid windows, number of window per level and every window coordinate.

- **get_scale_space_pyramid_images.m** - function for computing all the pyramid window images and its coordinates given an input image. Input: image to process Output: pyramid; cell array containing in a pyramid structure all the windows of the scale-space pyramid, coordinates; upper-left coordinates for each windows in the return pyramid structure.
- **get_window.m** - function for retrieving a specified sized window from an image. Input: image to process, width and height of the desired window and the extraction method. The possible method are centered or random, where the centered method extracts a centered window of the specified size from the input image, while the random method simply picks any valid window of the specified size somewhere over the input image. Giving a coordinate as a method leads on a window whose upper-left coordinate coincides with the specified one. Output: the extracted window
- **non_max_suppression.m** - function responsible of applying a suppression of all nearby detections selecting the most confident one. Two modes of proximity are defined. Windows not meeting a distance criterion or not meeting a threshold value of overlapping area are considered as nearby window and therefore likely to be compared to find the most confident between them. Input: window pixel coordinate, window confidence measure and window size. Output: Empty
- **plot_DETcurve.m** - function for plotting one or more overlapped DET curves given one or more SVM models. Input: List containing the models to test, cell array containing the model names as should be represented in the legend, positive and negative folder paths. If no positive nor negative paths are given a windows prompts for the folder paths. Output: Empty
- **plot_ROCcurve.m** - Script responsible of plotting the ROC curve. A window prompts for the SVM model file and the folders containing the positive and negative training images.
- **plot_tsne_map.m** - function to plot the t-SNE map given the image paths. In case no paths are given prompts an explorer window to locate the folders. Input: Cell array containing positive and negative image paths. Output: t-SNE plot

6. IMPLEMENTATION AND USAGE

- **test_svm.m** - function to test a given libSVM model. Given the paths where to find the positive and negative images, reads all the images, extracts the HoG descriptor of each one and classifies every image to compare with the Ground Truth class. Finally computes all the numeric measures explained in §5.2 Inputs: Model path, negative and positive image paths. In case no paths are given prompts an explorer window to locate the folders. Output: Struct containing all the numeric measures explained in §5.2
- **train_svm.m** - function to train a libSVM model. Finds the best possible model by reading the training configuration from a training parameter file where the kernel type, cost and gamma ranges are specified. After reading the configuration, all the images found in the given positive and negative folder paths are read and their HoGs are computed. With the ground-truth label plus the descriptor a training matrix is used to build the model. The model is then saved in the specified path. In addition, in the same path, a cross-validation logging file is written. Each line specifies the current kernel parameter configuration and the accuracy reached. Together with the logging file a cross-validation grid plot is saved in order to easily examine the parameter space. Inputs: model name, model save path and image paths, in case no paths are given a window prompts for them as in test_svm.m. Output: libSVM model struct.
- **plot_reject_curves.m** - function responsible of plotting accuracy measures in function of the classification threshold. Plots *recall*, *precision* and *F-score* measures. Input: libSVM model and input image folder. Output: Empty

6.1.1 Auxiliar functions

- **draw_curve.py / draw_grid.py** Both scripts are intended to draw same training plots as the ones provided by the cross-validation *MATLAB* function but solving some drawing issues that may occur when plotting axis values.
- **performance_to_excel.py** Script to parse test file information from a text file and write a excel worksheet with every test measure. This allows to easily make bar plots and compare different accuracy measures from one or more models together. Internally uses *workbook.py*, a class acting as an abstraction layer, to easily write tables in *excel sheets*.

6.2 Usage

A very simple interaction is needed to run any of the detectors as we provide a compiled version for every implemented one. The detectors consists in an *.exe* file, under *Windows* operating systems, so only double click is needed to launch them. Once the file es executed a console will show up and a window dialog will prompt for a folder containing the input images in *JPG* or *PNG* format. Once at a time the detector will perform all the operations and show the detections found as green boxes around the area where a pedestrian is detected together with a numeric measure representing the detection confidence. The application will wait for a key press, focusing the console window, before continuing with the next image. Figure 6.1 shows how it looks the detector application while running.

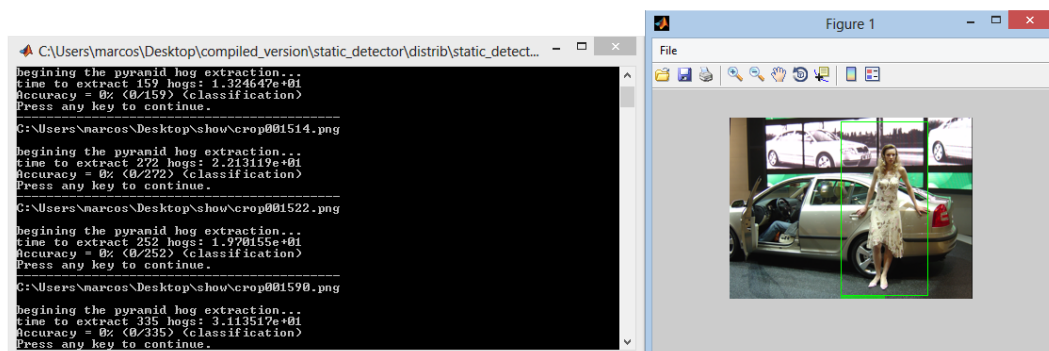


Figure 6.1: Static-detector compiled version - At the left the console shows how the application waits for a key to be pressed before processing the next image, at the right the woman is surrounded by a green bounding box indicating a pedestrian detection

6. IMPLEMENTATION AND USAGE

7

Performance

Until now we have completely covered the theoretical background and how to use this theory to build a working pedestrian classifier. Now we give some details on how the detectors perform. We do so by comparing all the models between them, so we end up with the most outperforming one. This is attained by doing to every final stage model all the performance test explained in §5.2.

In order to present information in the most compact way possible, instead of presenting every individual curve on its own we have grouped all the curves of the comparable models between them. In this chapter we will not present any ROC curve. Given the low error made by some models, ROC curves are not a good way to represent their behavior. The ROC curve gives nearly an area equals to 1 and becomes hard to spot differences when they become really subtle. Conversely we are providing DET curves for all the comparisons. Some other measures, like the mentioned ROC curves or the classifications scores distributions may be found attached in the Annex.

Is also important to note that the following tests were performed giving images of the window size to each model, so no window sliding nor scale-space pyramids intervened in the process. This must be known to understand the differences in performance that may occur when testing with the detectors, not on windows, but on full images.

7.1 MIT models

As the MIT models were found to be much easier to train and gave very good results, we begin with their performance measures. For the MIT data set only two models were

7. PERFORMANCE

trained due to the great success achieved in the first attempt. Only linear models were trained and only one round of re-training was needed to achieve perfect separation.

MIT linear pre models versus definitely MIT liner model.

With table 7.1 we try to illustrate the little difference a re-train makes when talking about the MIT person data set. As the first model trained already performed really well, obtaining any improvement in further re-training steps becomes nearly impossible. The only motivation to continue training is due to the fact that the first trained model made mistakes of classifying negative windows as positives, error that hopefully could be solved by adding more negative examples. Owing to the exceptional performance achieved by this models, also DET curves become unhelpful when trying to illustrate their differences. Consequently we opted to give some numeric measures.

Model	OKs	KOs	f_p	f_n	t_p	t_n	miss rate	Precision	Recall	F-score
pre model	725	3	3	0	274	451	0	0.9891	1	0.9945
def model	726	1	1	0	276	453	0	0.9963	1	0.9981
def model (opt th)	728	0	0	0	274	454	0	1	1	1

Table 7.1: MIT linear models performance comparison table

As can be seen in the MIT case nearly perfect separation is achieved, reaching a 99% of accuracy in the detection in both cases without searching for the optimum classification threshold, this is using a 0.5 default threshold. When using the optimum threshold found in figure 7.1, for the definitely model we achieve an absolutely perfect separation for the test set provided in the MIT data set.

MIT linear def model. RGB images versus gray scale images.

In figure 7.2 we study the impact of using images with only one color channel, this is gray scale images, when classifying with RGB trained models. As we can appreciate the performance is almost the same, therefore is not worth to train separate models for gray and color images as the detector is not so sensitive to the number of channels. Even though if possible is better to train and use RGB color images.

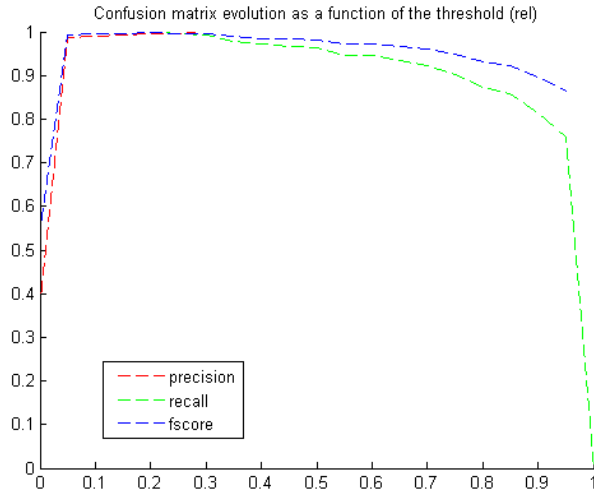


Figure 7.1: MIT linear def model threshold search - X axis represent threshold value, Y axis represents the performance measure value

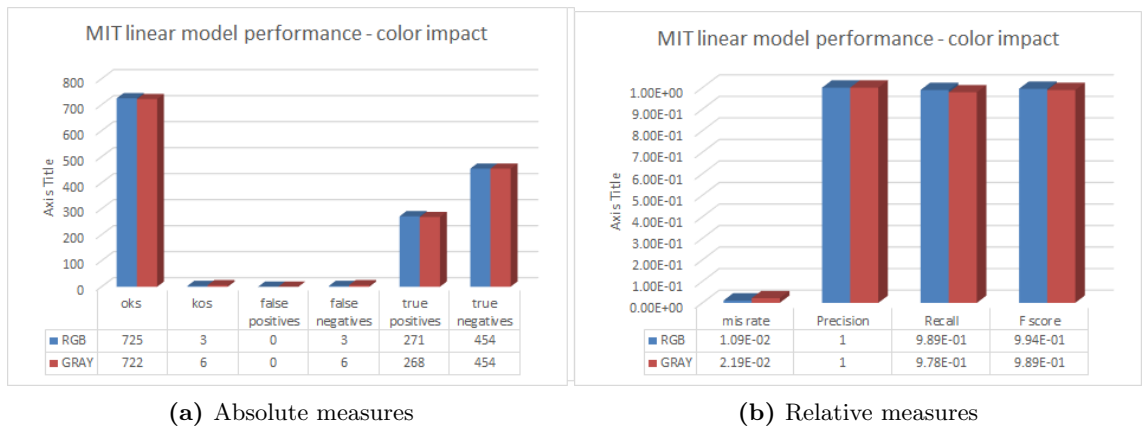


Figure 7.2: Color information impact in MIT linear model

7.2 INRIA models

As the INRIA data set contains a wider range of poses, partial occlusions and more illumination variances, happens to be much more complicated to end up with an SVM model as good as the MIT one. Hence we performed a broader training stage but also

7. PERFORMANCE

a broader testing stage, including a lot of comparisons between models to find the best possible options.

The INRIA person data set provides 2416 positive windows and 453 negative images to test. From the negative images, we randomly sampled 3 windows per image, obtaining 1359 negative windows. This was done like this to perform all the test without having skewed classes that could potentially distort some measure. More details of the image data sets can be found in §5.1.1.

As we saw in §5.1.2 different approaches were followed when training the INRIA models, depending on the ratio between positives and negatives instances. The first approach consisted in training every model with a similar amount of positives and negatives instances while in the second all the training instances were used. This is, using a data set where the available negative instances represent around 5 times the positive ones. For each approach linear and Gaussian kernels were trained. In the following section we present the performance measures grouped by approach followed and kernel type first and finally a review between the best models found in the previous comparisons in order to find the best option regarding the kernels.

7.2.1 1st training approach

7.2.1.1 Linear models

As regards to the first method and the linear kernels, we have that, as pointed by the cross validation curves showed in the SVM training chapter, little improvement is achieved by re-training with the hard examples images. The improvement achieved can be seen in the 7.11, showing how the optimal threshold value makes the second model slightly better.

Bar plots in figure 7.4 illustrate the little difference between both models. The middle bar represents the performance of the re-trained model when no optimal threshold is used, this is, when a 0.5 threshold determines the prediction class. In this case the performance falls dramatically.

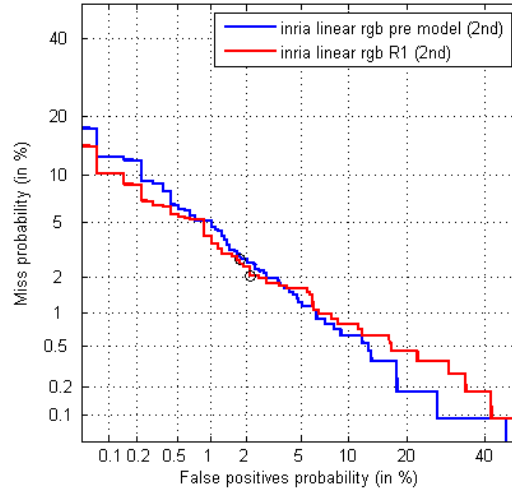
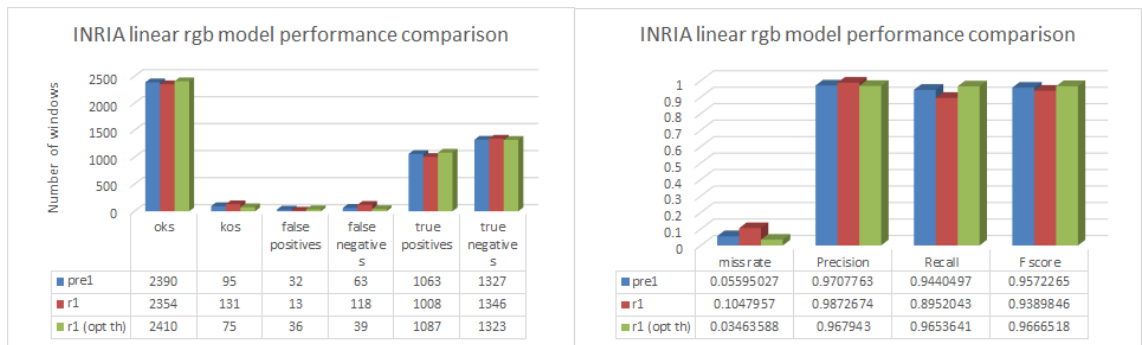


Figure 7.3: INRIA linear rgb pre-model versus round-1 model DET curve (1st approach) - The optimal threshold value yields a slightly better model after the re-training



(a) Absolute measures

(b) Relative measures

Figure 7.4: re-train impact in INRIA linear model (1st approach)

A fast review of the classification probabilities shows that the round-1 model is very confident in predicting negative instances, so a little suspicion about a particular window being a positive instance is enough to determine that is actually a positive one. That gives a optimal threshold of 0.1 for the positive prediction probability, what is the same as saying that, if the detector is not sure at at least at a 90% of not being a positive instance, then is not a positive instance. The optimal threshold search can be seen in figure 7.5.

7. PERFORMANCE

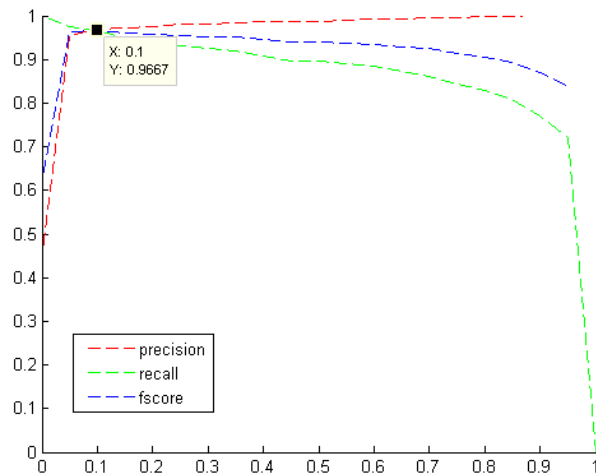


Figure 7.5: INRIA linear rgb round1 model threshold search (1st approach) - X in the figure shows optimal threshold, Y shows the convergence of the three performance measures

7.2.1.2 RBF models

Similarly as we did for the linear models, we compare now the performance for every Gaussian or RBF model. In this case more than one round is needed to find a model from which new steps of re-training does not improve. Anyhow the three first steps are essential as at every new re-train we reached a much better model. DET curves in figure 7.6 show the improvement of every model.

Concrete numeric measures over the test set can be seen in 7.7. As pointed out by the DET curves in 7.6, every stage improves all of the possible performance measures of the model.

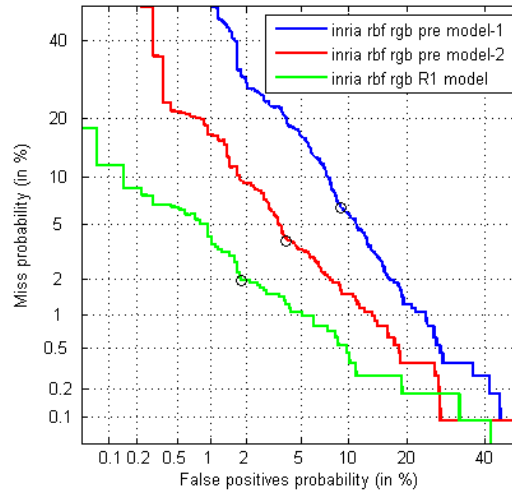


Figure 7.6: INRIA RBF rgb training rounds DET curves (1st approach) - At every re-training stage we achieve around a 50% of error reduction.

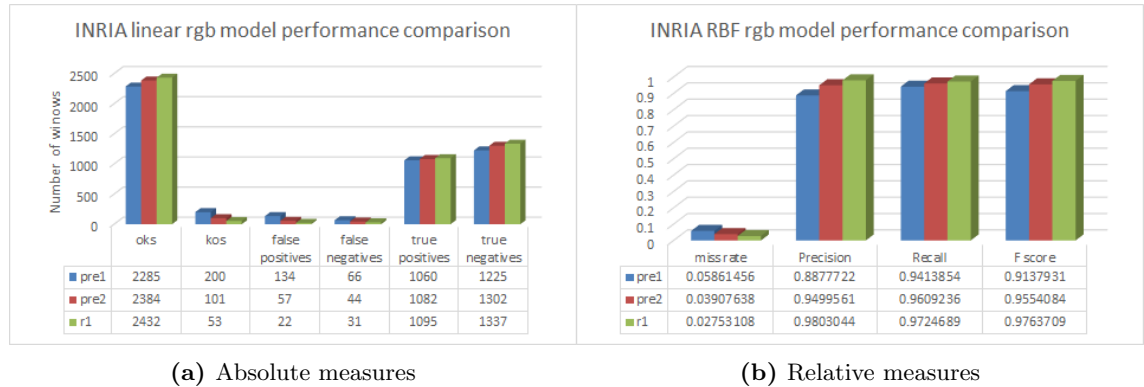


Figure 7.7: re-train impact in INRIA RBF models

It should be noted that these measures have been taken from the optimal threshold found by 7.8 where the optimality criterion is to minimize a cost function where false positives and false negatives penalize in the same ratio. In other words, maximize the F1-score. As explained earlier, a different weighting might be desired, for that purpose the same plots could be used.

7. PERFORMANCE

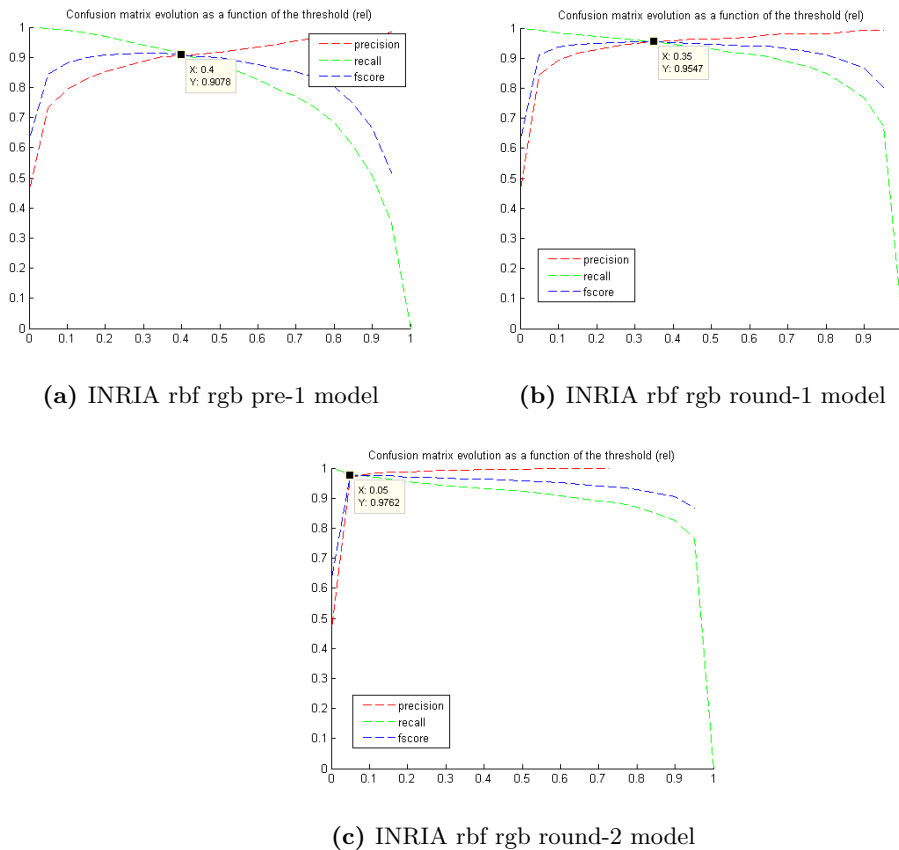


Figure 7.8: optimal threshold search for INRIA RBF models (1st approach)

7.2.1.3 Linear versus RBF model comparison

Now we have the best model for each kernel type, following the same measures, we compare the performance between the linear and the Gaussian or RBF models. As shown in 7.9 the Gaussian model performs better at every threshold level.

As we had a large amount of data to train and a rather complex feature descriptor we can take advantage from a more complex model that can generate more complex boundaries, therefore obtaining a better discrimination capability. Even though the RBF kernel performs better at every threshold, is noteworthy the effort needed to achieve this gain. In one hand because of the much more expensive cross-validation process, and in the other hand because of the extra re-training steps required to achieve the optimum model.

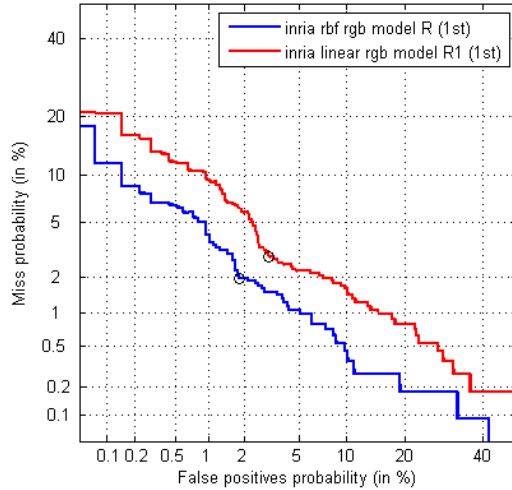


Figure 7.9: INRIA linear vs rbf models DET curves (1st approach) - DET curves show an advantage for the kernel model

The difference in performance when using optimal thresholds for both models can be seen in 7.10

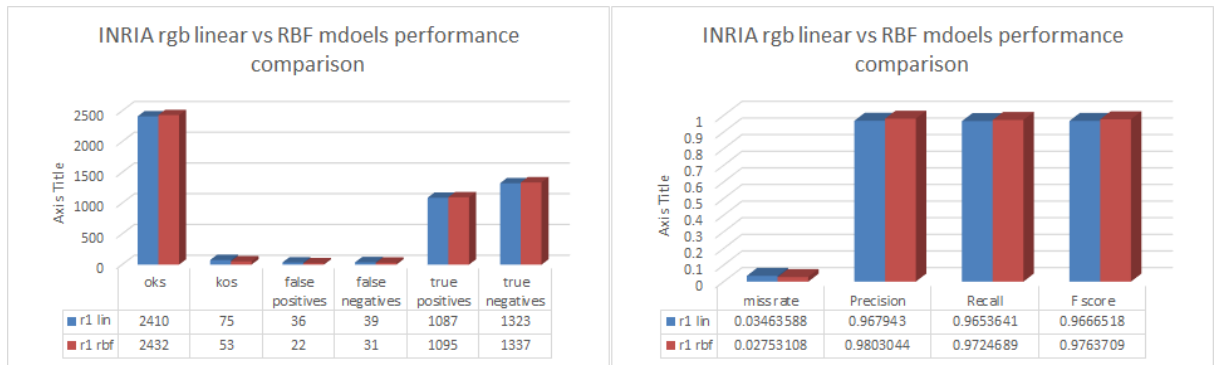


Figure 7.10: INRIA linear model versus RBF model performance comparison (1st approach) - The RBF model increases in about a 2% the precision and around a 1% the overall performance respect the linear one

7. PERFORMANCE

7.2.2 2nd training approach

7.2.2.1 Linear models

Same methodology as for the first approach follows. First we compare the performance between the different steps needed by the linear models to achieve the best possible results. As before this can be seen with the DET curves in figure 7.11

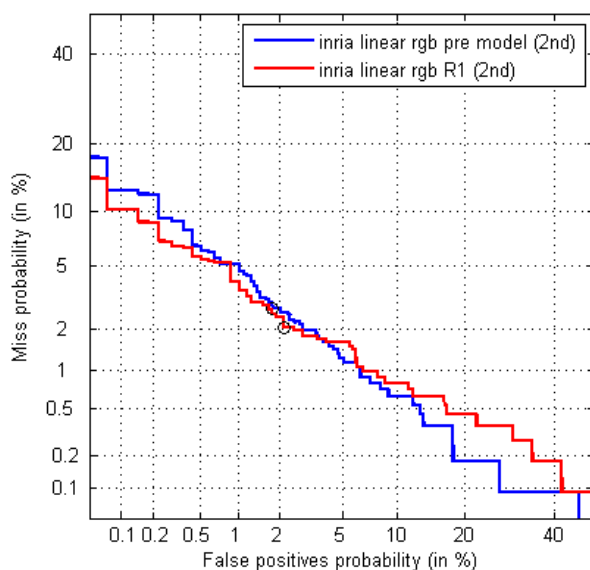


Figure 7.11: INRIA linear rgb per-model versus round-1 model - Round-1 re-training step improves the miss rate error

As shown in figure 7.11, is not so easy to determine which model performs better at a first glance. A numeric measure at the optimums thresholds reveals a slightly little gain for the round-1 model. Specially in figure 7.12 a subtle improvement can be seen in the miss rate measure in favor of the round-1 model.

7.2.2.2 RBF models

For the RBF models following the second approach only one train step was performed as further re-training did not improve. Anyway a hard parameter search was needed to find a good performance. In the following section all the figures will refer not to different re-training steps but to different parameterization that lead to quite different models. Even though we will not present the concrete performance measures for every

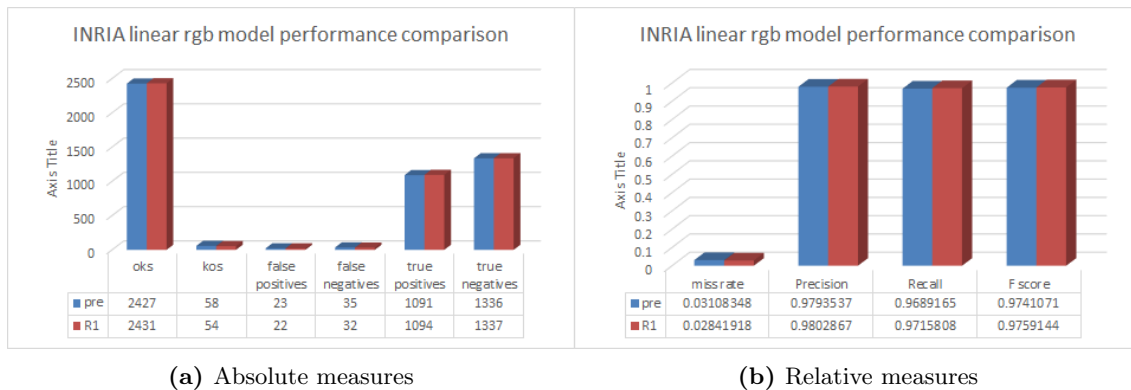


Figure 7.12: Re-train impact in INRIA linear model (2nd approach)

cross-validation search made while searching the best Gaussian model, we show in figure 7.13 the DET curves showing the evolution made by the performance of every model depending on the training parameters. Finally in figure 7.2 we show the numeric measures obtained with the final Gaussian model.

OKs	KOs	f_p	f_n	t_p	t_n	miss rate	Precision	Recall	F-score
2457	28	16	12	1114	1343	0.01065	0.9858	0.9893	0.9875

Table 7.2: INRIA RBF final model performance.

7.2.2.3 Linear versus RBF model comparison

Again, in order to find the best model from this training approach, we compare the performance over all the linear and RBF models. Figure 7.14 shows how the Gaussian model outperforms over the linear model, achieving around a 1% of improvement in both types of errors. This makes the Gaussian model, again, the best choice.

7.2.3 Selecting the final model

Once completed all the possible comparison over the whole set of models trained, we show a more general comparison from the model kernel type point of view. With this section we pretend not only to determine the best model between all the trained ones, but we want to know which type of approach proved to be the most successful one, which approach give the best models and with less effort.

7. PERFORMANCE

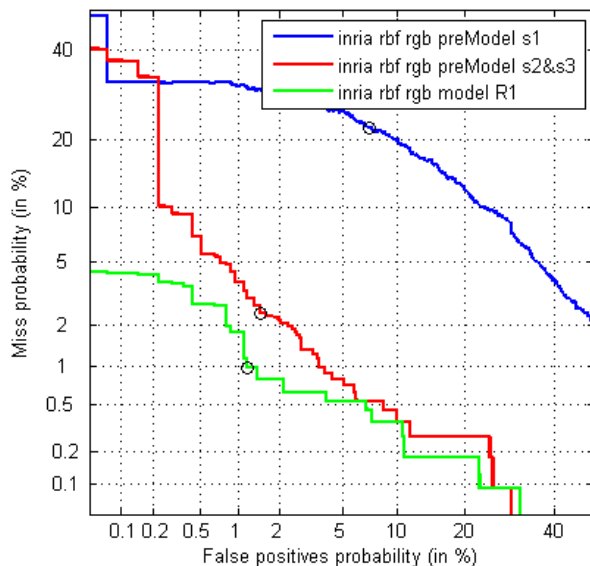
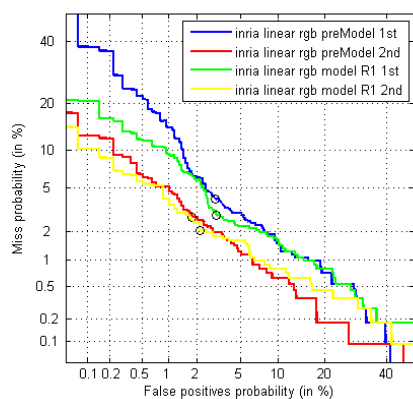
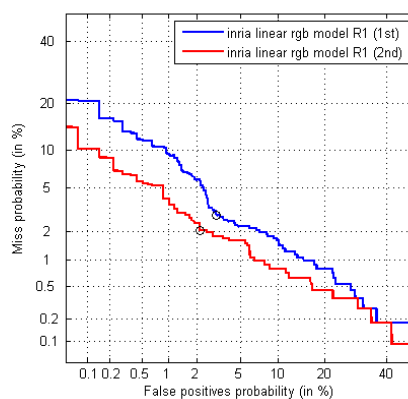


Figure 7.13: INRIA RBF model performance due to the parameter configuration - Every curve represents a parameter search from 1 to 3 until a outperforming model was achieved

Figure 7.15 illustrates the comparison between all the linear models on one hand and a comparison of the best linear models from each training approach.



(a) All linear models comparison



(b) Final linear models comparison

Figure 7.15: Linear models comparison between the 1st and 2nd training approach

As can be seen in figure 7.15, the final round-1 models and the pre-models in both cases

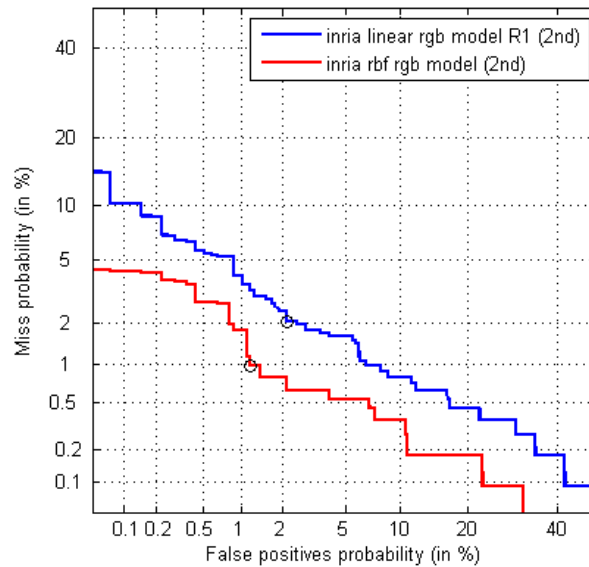


Figure 7.14: INRIA linear versus rbf model performance comparison (2nd approach) - DET curves show an advantage about a 1% for the RBF model

are better when trained following the second approach, this is, using all the available data. Is important to note that even the best 1st approach linear model is worst than the 2nd approach pre-model. Figure *b* isolates the best two linear models from each training approach, and clearly illustrates the improvement that involves the 2nd approach regarding the linear models.

The same analysis between the RBF models will lead us to the most outperforming model found in the previous sections so we can objectively conclude which SMV model will give us the best detection accuracy given our optimality criterion. Figure 7.16 shows how the best possible model turns out to be the RBF or Gaussian model trained following the second approach. The advantage is in decreasing both type of errors (false positives and false negatives) in around a 1%.

7. PERFORMANCE

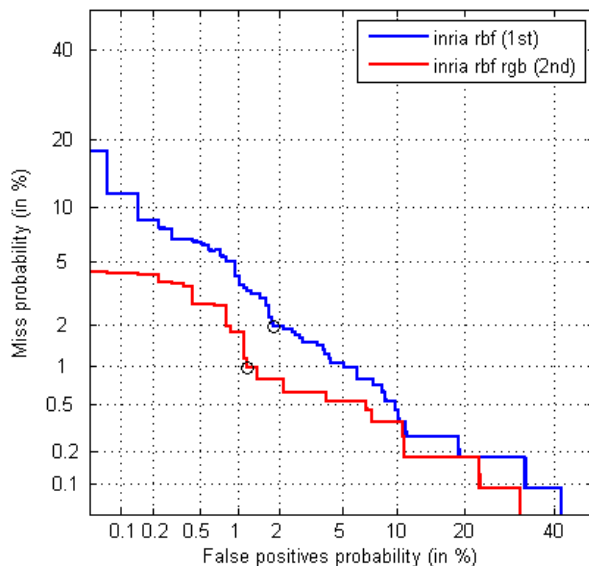


Figure 7.16: INRIA RBF model comparison between 1st and 2nd training approach - As this curves show, the RBF model trained following the 2nd approach stands out from the rest.

7.3 Detector performance

With the final model selected the only point to examine is how the detector performs w.r.t the time. Even though we provide a compiled version for every detector, the MATLAB tool¹ for obtaining a *.exe* file consist in packing every needed toolbox² together with the MATLAB Compiler Runtime (*MCR*)³ so no time improvement is achieved by this process. Anyway some independence from MATLAB is achieved as to run the application there is no need to have neither a MATLAB installation nor license.

Figure 7.17 show the time evolution w.r.t the input image size and the number of windows. Every measure has been done on a Intel i3 at 1.4 GHz with 4 Gb of RAM and Windows 8 (x64) operating system.

¹MATLAB compiler (mcc)

²Toolboxes are matlab packages providing extra algorithms and methods. For instance: Computer Vision Toolbox

³Standalone set of shared libraries that enables the execution of applications or MATLAB compiled components on computers that do not have MATLAB installed

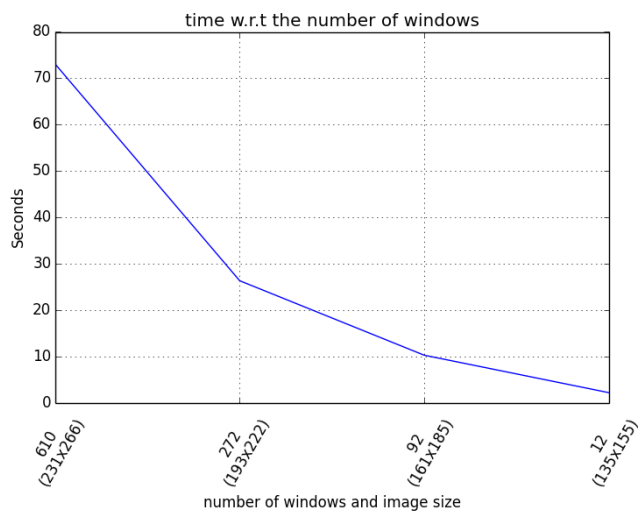


Figure 7.17: Detector time in seconds w.r.t the number of evaluated windows - Small increases in image size imply large increases in the number of windows and therefore in time.

Even though the HOG extraction process by itself achieves an average of 24 windows per second, when measuring the complete detector this throughput falls down to 8 ~ 9 windows per seconds. The scale-space-pyramid extraction together with the non-max-suppression steps have a serious performance impact.

To avoid this time issue a C translation was considered. We tried the *MATLAB Coder tool* (ccoder) but many code needed to be remodeled in order to make it suitable for the automatic code translation for the static detector version. Figure 7.18 show the MATLAB code analysis.

Therefore we consider this point for further and future work.

7. PERFORMANCE

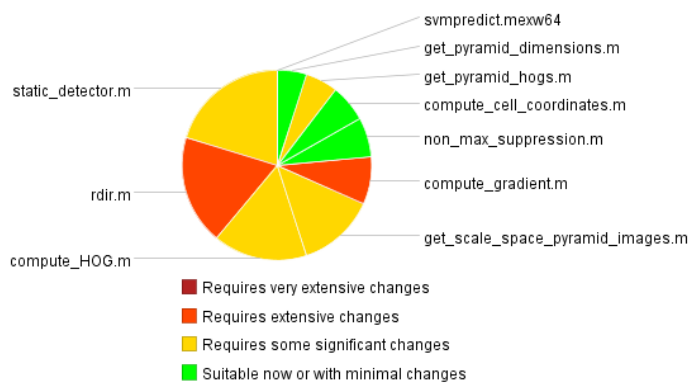


Figure 7.18: MATLAB code analysis for C code generation - Cell arrays and recursion is not yet supported by ccoder.

8

Discussion

8.1 Possible improvements

In the following subsections we review some possible improvements not implemented nor tried during the realization of this thesis. They are explained just to provide alternatives and further experimenting areas. Many of the improvement are focused on achieving a real-time detection. This is not an easy goal as many of the state-of-the-art pedestrian detectors use the sliding window paradigm.

8.1.1 Code compilation.

The first contribution to achieve some time improvement is to change the implementation language, so the next obvious step, if we were trying to develop an effective pedestrian detector application, will be to develop the project with all the found results in a more low level approach. At an early stage, where a large number of experiments and alternatives are considered, a fast and easy language as MATLAB turns out to be one of the most convenient options. When a more mature stage is reached other programming languages, more low level ones, should be the desired implementation option as we would obtain much more shorter execution times.

Even though a more low level implementation is completely out of the scope of this thesis, little research has been done in order to provide some recommendation on useful packages for a possible *C* or *C++* implementation. The SVM package used in this particular implementation (*LibSVM*) is available for many other languages, including *C++*. Specially for the *C* and *C++* languages *OpenCV* libraries provide very inter-

8. DISCUSSION

esting tools in order to easily manage images, deal with image filtering and much more image processing tasks.

8.1.2 Parallel GPGPU computation.

New-generation GPUs have a many-core architecture (hundreds of cores) and supports running thousands of threads in parallel, giving large memory bandwidth and very high performance, achieving magnitudes of 10^{12} *Floating Point Operations per second* (TFLOPS).

NVIDIA CUDA is an SDK, software stack and compiler that allows for the implementation of parallel programs in C for execution on the GPU.

As the HOG based detector is highly parallelizable, an interesting improvement could be achieved with GPGPU computation and the *NVIDIA CUDA* framework. Other frameworks can also be considered, *OpenCL* is also an interesting alternative.

As described in (1) a 67x improvement can be achieved by following this approach where every intensive computation is delegate to the GPU and only the non-maximum-suppression is done at the CPU.

The mentioned scheme is illustrated in figure 8.1

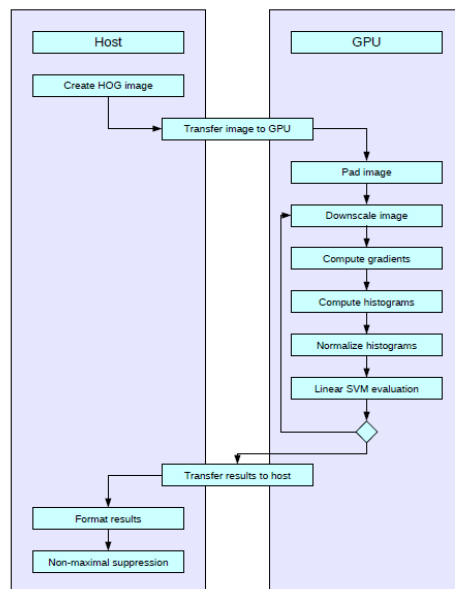


Figure 8.1: GPU implementation steps followed in (1) - Almost all the computation is delegated to the GPU achieving around 70x time improvement.

8.1.3 Principal Component Analysis for dimensional reduction.

Even though our default HOG feature vector has 3780 dimensions, using different overlapping, cell size, block size or number of bins will result in a variation in the dimension of the HOG feature vector. Dealing with larger vectors can increase the detection performance but at a expense of a higher time consumption. Therefore we are interested in a way of reducing high dimensional feature vectors in such a way that a good trade-off between time and performance loss is achieved. One possible way of dimensional reduction can be attained by using (*PCA*) over the descriptor matrix. The main idea is to apply a Singular Value Decomposition *SVD* to factorize the feature matrix at the training stage, capture the dimensions in which the most variation occurs and reduce the training matrix accordingly to only those dimensions. Geometrically we try to project

Formally: Let M_d be a $m \times n$ descriptor matrix, where each row is a HOG feature vector for a concrete instance. Being m the number of training instances and n the dimension of the feature vector. Let Σ be the co-variance matrix, defined as (with M_d):

$$\Sigma = \frac{1}{m} M_d^T M_d$$

Then the *SVD* factorizes the co-variance matrix as:

$$\Sigma = U S V^T$$

Where U is a matrix where each column is a left singular vector of Σ , S is a diagonal matrix with every singular value of Σ sorted in decreasing order. (Let's denote every element of S as λ_{ij}). As S is a diagonal matrix we have that $\lambda_{ij} = 0$ for every pair of i and j such that $i \neq j$. On the other hand V is a matrix with each column being right singular vectors of Σ . We want to find the k principal components of our feature matrix retaining a concrete amount of variance (σ) in our data. In order to achieve such variance retention we must find k in such a way that:

$$\frac{\sum_{i=1}^k \lambda_i}{\sum_{i=1}^n \lambda_i} > \sigma$$

Keeping the k first vectors of U ($U_{reduced}$) we obtain a reduced descriptor matrix as:

$$M_{d(reduced)} = U_{reduced}^T M_d$$

8. DISCUSSION

Therefore *SVM* will deal with shorter feature vectors, what it is translated in less computational effort and time.

Our brief tests on applying *PCA* to the HOG feature matrix show that it is possible to reduce a 3780 dimensional HOG to a 937 dimensional one by sacrificing in more or less a 1% the overall performance. However we must take into account that only a pre-model has been trained following this approach (so no re-training steps) and no optimal threshold has been searched.

Figure 8.2 show the cross-validation curve and table 8.1 show the test measures applied to a linear model trained with reduced HOGs.

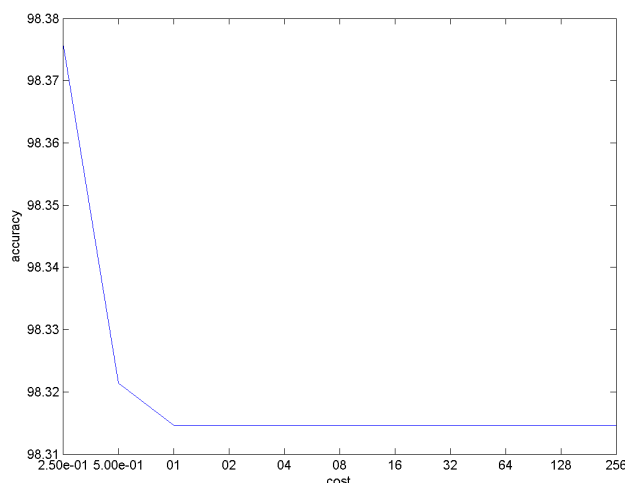


Figure 8.2: INRIA linear PCA-model cross-validation curve - Cross-validation of a linear model trained with PCA-reduced features.

OKs	KOs	f_p	f_n	t_p	t_n	miss rate	Precision	Recall	F-score
2410	75	55	20	1106	1304	0.0176	0.952	0.982	0.967

Table 8.1: INRIA linear PCA-model performance.

Anyway, as already mentioned, further test should be done in order to obtain a more accurate conclusion, optimum reductions and maybe possible drawbacks not noticed during our brief testing.

8.2 Alternatives

At the beginning of this thesis we have mentioned some methods developed by many different institutes and investigators apart from the developed one. Anyway all the commented methods consist in defining some kind of feature and then classifying them following different scanning approaches and trying many of the possible enhancements like early rejections etc. Recently some interesting results have been found using *Artificial Neural Networks* (ANN), specially *Convolutional Neural Networks* (CNN). Surprising results were achieved in many different fields and problems; Handwritten digit recognition (23), object recognition (24) or autonomous driving. Concretely the work developed in (24) by Alex Krizhevsky achieved a much lower error rate in the ILSVRC-2012 competition compared with many famous computer vision groups all over the world. Among several advantages, the CNN integrates feature extraction and classification into one single, fully adaptive structure. Furthermore we do not need to worry about defining complex features representations as the embedding layers of the network already define implicitly what features best represent the inputs. They are designed to recognize visual patterns directly from pixel images with minimal pre-processing. Another significant improvement has to do with the relative tolerance that CNN show in front geometric transformations, local distortions etc.

Figure 8.3 show a representation of the features learned by a hidden layer of 25 hidden units on an ANN designed to recognize handwritten digits. The inputs for the network were 20×20 images with a handwritten digit. After the training stage every hidden unit had a most likely activation representation. This ANN was just a little example with only one input layer, one hidden layer and one output layer.

Anyway, defining the topology of deep networks and training them can be a very expensive task, so, many resources may be needed to find good results. In order to illustrate this idea we give some numbers of the deep net trained in (24):

- Trained with stochastic gradient descent on two *NVIDIA GPUs* for about a week
- 650,000 neurons
- 60,000,000 parameters
- 630,000,000 connections

8. DISCUSSION

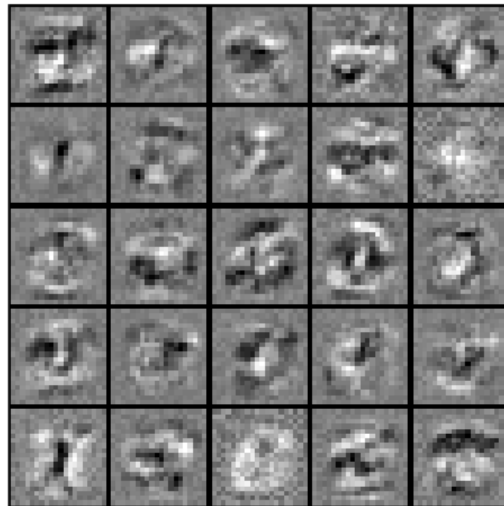


Figure 8.3: Hidden layer feature representation learned by a simple ANN - Every square image represents the inputs that make a concrete neuron to be activated.

- Final feature layer: 4096-dimensional

Even though it could be a very interesting approach for further investigation, Y. Bengio and Y. LeCun present in (25) theoretical and empirical evidence showing that kernel methods and other "shallow" architectures are inefficient for representing complex functions such as the ones involved in artificially intelligent behavior, such as visual perception.

9

Materials & methods

9.1 MATLAB

9.1.1 variable types

Even though the aim of this chapter is not making a complete explanation about the *MATLAB* variables, we give a brief description about some data types used in the project implementation.

Mainly *MATLAB* works with matrices of different numeric representation, anyway non numeric types can also be stored in a matrix while some requirements are fulfilled. Figure 9.1 show the possible data types that can be stored in a matrix form. The only requirement that a matrix must meet is to have all dimension being consistent, this is to say, all elements in a matrix must have the same dimension and all elements within the matrix must of the same type.

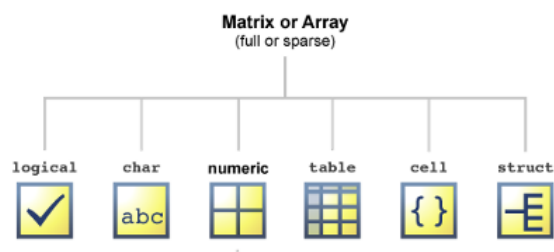


Figure 9.1: *MATLAB* matrix data types - For a more accurate type explanation refer to the *MATLAB* documentation in (26)

9. MATERIALS & METHODS

Cell Array.

For storing some varying size data, lets say, strings of different size in a vector or array, *MATLAB* provides an special variable type called cell arrays which is a data type with indexed data containers called cells, where each cell can contain any type of data. Cell arrays commonly contain either lists of text strings, combinations of text and numbers, or numeric arrays of different sizes.

Mat files.

Files with a *.mat* extension contain *MATLAB* formatted data. This data can be loaded from or written to these files using the functions *LOAD* and *SAVE*, respectively. We use this format to store the *SVM* models and the different parameters needed to configure the scale-space-pyramid procedure, testing or training process etc. More information about the *.mat* files is available in (27).

9.2 Libraries and packages

9.2.1 libSVM

Many SVM packages are available but *libSVM* turns out to be one of the most popular and complete ones. It is an open source library developed at the *National Taiwan University*, is developed in *C++* and supports classification and regression. It is free software published under a *BSD* license. Among its advantages we may notice that counts, trough many interfaces, with compatibility to a wide range of languages and platforms, this is noteworthy because just in case the code is ported to another language the same classifier with same parameters and configuration could be used.

9.2.2 NIST DET plots

In order to plot *Detection Error Tradeoff* curves a *MATLAB* package provided by the National Institute of Standards and Technology (*NIST*) is used. The complete software can be downloaded in (28). For additional information about DET curves refer to (29).

9.2.3 m2html

With the purpose of delivering a comprehensive code documentation in a easy to read and ecologic way, all the *MATLAB* code has been parsed with the *m2html* functions

creating a set of *HTML* pages containing all the information regarding the code. In this way we explicitly define the relationship between functions, inputs, outputs and the code itself while making extremely easy navigating through all the source code and files. The tool itself and all the information about it can be found in: (30).

9. MATERIALS & METHODS

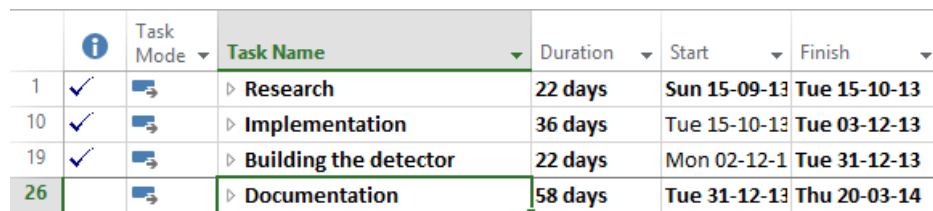
10

Project Management

10.1 Planning

The project is divided in four main sections; Research, Implementation, Building and Documentation. The research step consists in the information compilation and understanding, the implementation considers all the code developing, the Building stage collects all the model training, testing and selection and finally the Documentation phase has to do with all the results recollection, and documentation writing.

Figure 10.1 shows the overall project planning.








		Task Mode ▾	Task Name ▾	Duration ▾	Start ▾	Finish ▾
1	✓		▷ Research	22 days	Sun 15-09-13	Tue 15-10-13
10	✓		▷ Implementation	36 days	Tue 15-10-13	Tue 03-12-13
19	✓		▷ Building the detector	22 days	Mon 02-12-13	Tue 31-12-13
26			▷ Documentation	58 days	Tue 31-12-13	Thu 20-03-14

Figure 10.1: Overall project schedule - Collapsed view of all four project phases

A more detailed view of the research step reveals two main stages; the first one is an approach to the theory behind the HOG descriptor, SVMs and the pedestrian detection problem. The second one has to do with the selection of the developing environment w.r.t the programming language, tools, libraries and available information or community. Figure 10.2 shows the original schedule. The State-of-the-art step took longer because of the need of further research as other pedestrian detection alternatives involved some unseen theory.

10. PROJECT MANAGEMENT

		Task Mode ▾	Task Name ▾	Duration ▾	Start ▾	Finish ▾
1	✓		▾ Research	22 days	Sun 15-09-13	Tue 15-10-13
2	✓		▾ Pedestrian detection	22 days	Sun 15-09-13	Tue 15-10-13
3	✓		HOG descriptor	7 days	Sun 15-09-13	Mon 23-09-13
4	✓		SVM	7 days	Mon 23-09-13	Tue 01-10-13
5	✓		State-Of-The-Art	7 days	Tue 01-10-13	Wed 09-10-13
6	✓		▾ Languages and environments	5 days	Wed 09-10-13	Tue 15-10-13
7	✓		MATLAB	2 days	Wed 09-10-13	Thu 10-10-13
8	✓		Python	2 days	Wed 09-10-13	Thu 10-10-13
9	✓		libSVM	3 days	Fri 11-10-13	Tue 15-10-13
10	✓		▸ Implementation	36 days	Tue 15-10-13	Tue 03-12-13
19	✓		▸ Building the detector	22 days	Mon 02-12-13	Tue 31-12-13
26			▸ Documentation	58 days	Tue 31-12-13	Thu 20-03-14

Figure 10.2: Research schedule - Detailed view which comprehends a theory and technology study

Once all the theory was digested and MATLAB was chosen the implementation phase began, developing first the HOG feature extraction and followed by the classifier construction. Also all the measurement tools were developed in order to be able to parallelize the training, testing and evaluation of the models. Figure 10.3 shows this breakdown. Gratefully some pre-build functions for plotting measures like ROC curves and visualizing HOGs helped to reduce the data visualization phase span.

The next stage consists in training and testing the SVM models. When possible the training and testing stages were overlapped in time, using two different computers. Due to the long computational times needed on order to train complex models like RBF ones with a large data-set and wide parameter-spaces cross-validations, the model building phase was delayed. Mean while the final detector developing was carried out. Figure 10.4 shows this temporal overlapping.

Finally the results collecting and documentation writing stage compiled all the project explanation. As the code was properly documented in a first stage the only pending work was to make a easy to use and understand documentation. Using a matlab to html converter the code explanation turned out to be very friendly in effort and time. It is noteworthy that the redaction of the thesis is ,by itself, the largest step. Figure 10.5 show this detail.

		Task Mode	Task Name	Duration	Start	Finish
1	✓		▸ Research	22 days	Sun 15-09-13	Tue 15-10-13
10	✓		▸ Implementation	36 days	Tue 15-10-13	Tue 03-12-13
11	✓		HOG descriptor	7 days	Tue 15-10-13	Wed 23-10-13
12	✓		SVM trainer	10 days	Wed 23-10-13	Tue 05-11-13
13	✓		SVM tester	7 days	Thu 07-11-13	Fri 15-11-13
14	✓		▸ Evaluation functions	8 days	Sat 16-11-13	Wed 27-11-13
15	✓		data visualization	5 days	Sat 16-11-13	Thu 21-11-13
16	✓		numeric measures	2 days	Thu 21-11-13	Fri 22-11-13
17	✓		graphic measures	4 days	Sat 23-11-13	Wed 27-11-13
18	✓		Review and vectorization	4 days	Thu 28-11-13	Tue 03-12-13
19	✓		▸ Building the detector	22 days	Mon 02-12-13	Tue 31-12-13
26			▸ Documentation	58 days	Tue 31-12-13	Thu 20-03-14

Figure 10.3: Implementation schedule - Detailed view showing all the implementation phase

		Task Mode	Task Name	Duration	Start	Finish
1	✓		▸ Research	22 days	Sun 15-09-13	Tue 15-10-13
10	✓		▸ Implementation	36 days	Tue 15-10-13	Tue 03-12-13
19	✓		▸ Building the detector	22 days	Mon 02-12-13	Tue 31-12-13
20	✓		model training	14 days	Mon 02-12-13	Thu 19-12-13
21	✓		models testings	13 days	Wed 11-12-13	Fri 27-12-13
22	✓		final model selection	7 days	Thu 19-12-13	Fri 27-12-13
23	✓		static detector implementation	2 days	Fri 27-12-13	Mon 30-12-13
24	✓		▸ Deployment	2 days	Mon 30-12-13	Tue 31-12-13
25	✓		static detector compilation	2 days	Mon 30-12-13	Tue 31-12-13
26			▸ Documentation	58 days	Tue 31-12-13	Thu 20-03-14

Figure 10.4: Building schedule - Detailed view of the building stage

10. PROJECT MANAGEMENT

		Task Mode	Task Name	Duration	Start	Finish
1	✓		▷ Research	22 days	Sun 15-09-13	Tue 15-10-13
10	✓		▷ Implementation	36 days	Tue 15-10-13	Tue 03-12-13
19	✓		▷ Building the detector	22 days	Mon 02-12-13	Tue 31-12-13
26			◀ Documentation	58 days	Tue 31-12-13	Thu 20-03-14
27			Code documentation	2 days	Tue 31-12-13	Wed 01-01-14
28	✓		Thesis	52 days	Wed 08-01-14	Thu 20-03-14
29	✓		Demos	6 days	Wed 01-01-14	Wed 08-01-14

Figure 10.5: Building schedule - Unfolded view of the building section

In general there has been no severe deviation from the time schedule even though tasks like the detector building or information searches about the State-of-the-art techniques turned out to be more time-consuming. Anyhow every deviation proved to be in such a way that little rearrangements could easily solve any mishap, commonly by using the time to compute expensive tasks to later on dispose of needed data like HOGs feature matrix. Figure 10.6 shows a complete view of the planning.

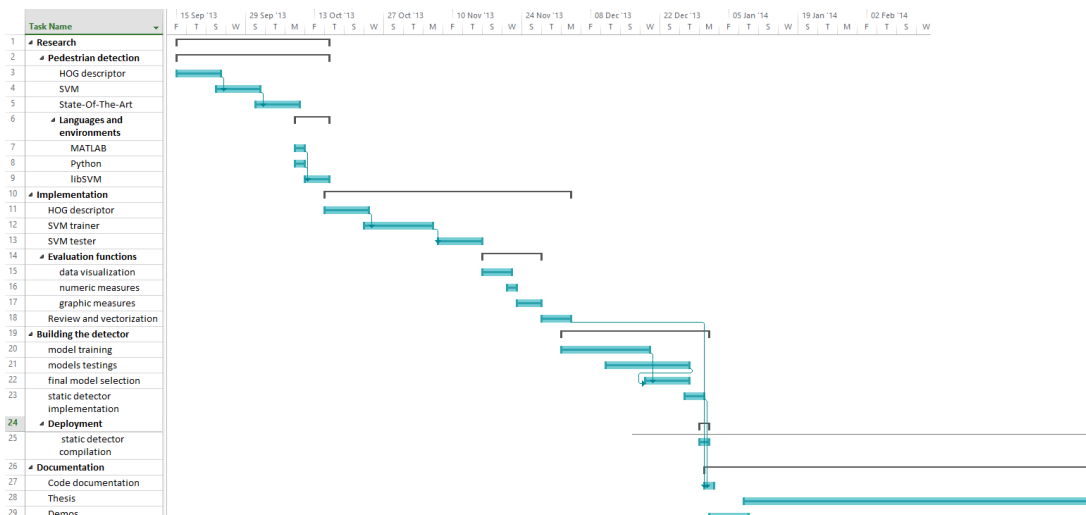


Figure 10.6: Complete schedule - Complete schedule and dependencies between tasks

10.2 Costs

The next sections exposes the associated cost due to the project development. We assume an average of 6.5 hours of work per day. We divide the cost explanation in three parts; Human resources, Hardware and Software costs.

w

10.2.1 Human resources

In order to compute the human resources costs, we average the hours of work of every stage. In order to take into account the difference effort needs of every phase, a weighing is applied through a estimation of the time load. Table 10.1 resumes this information.

Concept	Days	Hours	Time load	Price	Cost
Research	22	143	90%	35€	5005€
Implementation	36	234	100%	50€	11700€
Building the detector	22	143	60%	50€	1100€
Documentation	58	377	80%	25€	1450€
Total	138	897			19255€

Table 10.1: Human resources cost breakdown

10.2.2 Hardware resources

The hardware needed comprises:

- Laptop with Intel i3 quad Core processor at 1.4GHz with 4Gb of RAM memory ($\simeq 500$ €)
- PC with an Intel Core 2 6400 at 2.13GHz with 4Gb of RAM memory ($\simeq 400$ €)

Both computers sum up to around 900€, nevertheless both computers are not only being used for this project so we must take in consideration the hardware amortization. Considering three years like the average life for a computer:

$$\text{Hardware cost} = \frac{900\text{€}}{3 \text{ years} \cdot 12 \text{ months/year} \cdot 22 \text{ days/month} \cdot 8 \text{ hours/day}} \simeq 0.14\text{€/hour}$$

Even though table 10.1 show the total working hours, does not reflects the total amount of computing time,as for instance, the model computing and optimization stage that

10. PROJECT MANAGEMENT

required many days of computation without stop. To obtain a fair measure of the total computing time, we estimate a mean of 20 hours of computing time for the model building stage. Therefore:

$$\text{total computing hours} = 116 \text{ days} \cdot 6.5 \text{ hours} + 22 \text{ days} \cdot 20 \text{ hours} = 1194 \text{ hours}$$

Resulting in a hardware cost of:

$$\text{Hardware cost} = 1194 \text{ hours} \cdot 0.14\text{€/hour} = 167.16\text{€}$$

10.2.3 Software cost

Table 10.2 specifies the software prices excluding open source software like libSVM, Python, matplotlib etc. Therefore mainly MATLAB and Microsoft software comprises the payment software.

Software	Price
MATLAB R2014a	2000€
Image processing Toolbox	1000€
Computer Vision System ToolBox	1250€
Microsoft Excel 2013	135€
Microsoft Project 2013	769€
Total	5154 €

Table 10.2: Software costs

Applying a similar amortization formulation as before, assuming a complete MATLAB use during the whole project, only 8 hours for both Microsoft Excel and Project and a useful life for software around 3 years:

$$\text{MATLAB cost} = 1194 \text{ hours} \cdot 0.67 \text{ €/hour} = 2850.77\text{€}$$

$$\text{Microsoft Excel cost} = 8 \text{ hours} \cdot 0.0.21 \text{ €/hour} = 0.17\text{€}$$

$$\text{Microsoft Project cost} = 8 \text{ hours} \cdot 0.14 \text{ €/hour} = 1\text{€}$$

$$\text{Total} = 2851.91\text{€}$$

Finally the overall project cost is:

Project	Price
Human resources	19255 €
Hardware	167.16 €
Software	2851.91 €
Total	22274.07 €

Table 10.3: Total project cost

10. PROJECT MANAGEMENT

References

- [1] VICTOR ADRIAN PRISACARIU AND IAN REID. **fastHOG - a real-time GPU implementation of HOG**. Technical report, University of Oxford, Department of Engineering Science, 2009. ix, 76
- [2] **A trainable system for object detection**. *Intl. Journal of Computer Vision*, 2000. 4
- [3] P. A. VIOLA AND M. J. JONES. **Robust real-time face det.** *Intl. Journal of Computer Vision*, 2004. 4
- [4] M. YEH Q. ZHU, S. AVIDAN AND K. CHENG. **Fast human detection using a cascade of histograms of oriented gradients**. *IEEE Conf. Computer Vision and Pattern Recognition*, 2006. 4
- [5] F. M. PORIKLI. **Integral histogram: A fast way to extract histograms in cartesian spaces**. *IEEE Conf. Computer Vision and Pattern Recognition*, 2005. 4
- [6] P. PERONA P. DOLLAR, Z. TU AND S. BELONGIE. **Integral channel features**. *British Machine Vision Conf.*, 2009. 4
- [7] S. BELONGIE P. DOLLAR AND P. PERONA. **The fastest pedestrian detector in the west**. *British Machine Vision Conf.*, 2010. 4
- [8] N. DALAL I B. TRIGGS. **Histograms of oriented gradients for human detection**. *IEEE Conf. Computer Vision and Pattern Recognition*, 2005. 9, 13, 15, 16, 35
- [9] W. T. FREEMAN AND M. ROTH. **Orientation histograms for hand gesture recognition**. *Intl. Workshop on Automatic Face and Gesture- Recognition, IEEE Computer Society, Zurich, Switzerland*, 1995. 9
- [10] J. OHTA W. T. FREEMAN, K. TANAKA AND K. KYUMA. **Computer vision for computer games**. *2nd International Conference on Automatic Face and Gesture Recognition, Killington*, 1996. 9
- [11] D. G. LOWEA. **Distinctive image features from scale-invariant keypoints**. 2004. 9
- [12] J. MALIK S. BELONGIE AND J. PUZICHA. **Matching shapes**. *The 8th ICCV, Vancouver, Canada*, 2001. 9
- [13] **Sobel operator**. http://en.wikipedia.org/wiki/Sobel_operator. 13
- [14] **Gaussian smoothing**. http://en.wikipedia.org/wiki/Gaussian_blur. 13
- [15] L. MAATEN AND G. HINTON. **Visualizing Data using t-SNE**. *Journal of Machine Learning Research* 9, 2008. 17
- [16] **Kullback-Leibler divergence**. http://en.wikipedia.org/wiki/Kullback-Leibler_divergence. 17
- [17] TOMASZ MALISIEWICZ ANTONIO TORRALBA CARL VONDRICK, ADITYA KHOSLA. **HOGgles: Visualizing Object Detection Features**. 2013. 18
- [18] DMITRIY FRADKIN AND ILYA MUCHNIK. **Support Vector Machines for Classification**. N/D. 23
- [19] TRISTAN FLETCHER. **Support Vector Machines Explained**. 2009. 23
- [20] COLIN CAMPBELL KRISTIN P. BENNETT. **Support Vector Machines: Hype or Hallelujah?** N/D. 23
- [21] **MIT pedestrian dataset**. <http://cbcl.mit.edu/projects/cbcl/software-datasets/PeopleDataReadme.html>. 35
- [22] **INRIA person dataset**. <http://pascal.inrialpes.fr/data/human/>. 35
- [23] L. BOTTOU C. CORTES J. S. DENKER H. DRUCKER I. GUYON U. A. MULLER E. SACKINGER P. SIMARD Y. LECUN, L. D. JACKEL AND V. VAPNIK. **Learning Algorithms For Classification: A Comparison On Handwritten Digit Recognition**. 79
- [24] GEOFFREY HINTON ALEX KRIZHEVSKY, ILYA SUTSKEVER. **ImageNet Classification with Deep Convolutional Neural Networks**. 2012. 79
- [25] YOSHUA BENGIO AND YANN LECUN. **Scaling Learning Algorithms towards AI**. 2007. 80
- [26] **Matlab data types**. http://www.mathworks.es/es/help/matlab/data-types_data-types.html. 81
- [27] **Matlab .mat files**. http://www.mathworks.com/help/pdf_doc/matlab/matfile_format.pdf. 82
- [28] **NIST Evaluation tools**. <http://www.itl.nist.gov/iad/mig/tools/>. 82
- [29] T. KAMM M. ORDOWSKI M. PRZYBOCKI A. MARTIN, G. DODDINGTON. **The DET curve in assessment of detection task performance**. 82
- [30] **m2html tool**. <http://www.artefact.tk/software/matlab/m2html/>. 83