Paris-Dauphine University - EISTI
Image & Pervasive Access Lab

Master Thesis

Deep Learning: Solving the detection problem

Master Student:
Anne MORVAN

Supervisor at IPAL:
Dr. Antoine VEILLARD

A thesis submitted in fulfilment of the requirements
for the degree of Master of Science

named

"Informatics: Intelligent Systems"
of MIDO Department in Paris-Dauphine University

September 2015

PARIS-DAUPHINE UNIVERSITY - EISTI

Abstract

IPAL
Deep Learning Team

Master of Science

Deep Learning: Solving the detection problem

by Anne MORVAN

Deep learning algorithms such as convolutional neural networks dramatically changed the computer vision landscape by outperforming other state-of-the-art models in many object recognition tasks. Most benchmarks so far take place in the classification context, where an image known to contain a relevant object has to be labeled using one of the known object classes. In comparison, the use of deep learning algorithms in the object detection task is much less investigated.

During this internship, several aspects related to object detection have been examined with a particular focus on pedestrian detection. First, a state of the art is made on object and pedestrian detection. Then, a classifier model is proposed and the results for the pedestrian classification tasks are presented.

Key words: deep learning, classification, pedestrian detection, convolutional networks

Acknowledgements

I wish to thank everyone who contributed to the success of my internship and that helped me during the writing of this report.

First of all, my thanks to my teachers, Pr. Suzanne PINSON and Pr. Antoine CORNUEJOLS of Paris-Dauphine University, and Pr. Maria MALEK from EISTI (*Ecole Internationale des Sciences du Traitement de l'Information*) who enabled me to apply to this internship thanks to their recommendation letters.

I would like then to thank my internship advisors, Dr. Antoine VEILLARD and Mr. Olivier MORERE, respectively post-doctoral and PhD student at IPAL, for their welcome, the time they spent helping me and their expertise sharing. I also thank Dr. Vijay CHANDRASEKHAR, Dr. Hanlin GOH, Dr. Lin JIE and Ms. Julie PETTA who are the other members of the Deep Learning team.

Finally, I want to thank the whole team IPAL for their welcome.

Contents

List of Figures

List of Tables

Abbreviations

av.	**av**erage
bb	**b**ounding **b**ox
CNN(s)	**C**onvolutional **N**eural **N**etwork(s)
ConvNet	**Conv**olutional **Net**work
CPU	**C**entral **P**rocessing **U**nit
CUDA	**C**ompute **U**nified **D**evice **A**rchitecture
GPU	**G**raphics **P**rocessing **U**nit
HOG	**H**istograms of **O**riented **G**radients
IPAL	**I**mage & **P**ervasive **A**ccess **L**ab
JSON	**J**ava **S**cript **O**bject **N**otation
lr	learning **r**ate
neg.	**neg**ative
PCA	**P**rincipal **C**omposent **A**nalysis
pos.	**pos**itive
PReLU	**P**arametric **Re**ctified **L**inear **U**nit
ReLU	**Re**ctified **L**inear **U**nit
RGB	**R**ed **G**reen **B**lue
SGD	**S**tochastic **G**radient **D**escent
SIFT	**S**cale **I**nvariant **F**eature **T**ransform
std	**st**andart **d**eviation
SVM	**S**upport **V**ector **M**achine
UDN	**U**nified **D**eep **N**et

Chapter 1

Introduction

1.1 Motivation

The human visual system recognizes and localizes objects without any problem even within overfull scenes. For artificial systems, however, this is still a hard task because of two factors: the viewpoint-dependent object variability and the high in-class variability of many object types. For the pedestrian detection case, this intra-class variability includes color or kind of clothing, pose, appearance, partial occlusions, illumination or background.

Therefore one goal of researchers working in computer vision and artificial intelligence has been to give computers the ability to "see" or make visual analysis and interpretation of images or videos. For that matter, deep hierarchical neural models approximately imitate the functioning of mammalian visual cortex, and are commonly thought as among the most promising architectures for such tasks.

Besides, pedestrian detection is one of the most studied problems in computer vision in the past decade because of its numerous applications in automotive driving safety (smart cars), robotics and video surveillance. But yet only few deep neural networks have been applied to this task, though. Thus in this work we aim in particular to apply deep convolutional neural networks on this challenging problem in order to improve the current state-of-the-art results.

1.2 Raised issues

Computer vision and pedestrian detection in particular is not without difficulties. The following points are the most important problems to deal with in the context of (1) pedestrian detection and (2) deep learning.

1.2.1 Issues related with pedestrian detection task

Find good features The first step in pedestrian detection is to define good pedestrian detectors (or features) to enable the network to recognize one instance of pedestrian. As a reminder, the difficulty is to deal with the wide variety of appearances of pedestrians due to body pose, occlusions, clothing, lighting, and backgrounds which should not interfere in the detection process. Indeed, here two examples to illustrate the fact :

- the clothes color does not give relevant information for pedestrian recognition because everybody wear something different.
- the background should not influence the model, otherwise it can predict a pedestrian presence only because the environment is urban and not a forest.

Those subtleties should be taken into account in the features extracting process for representing efficiently the image and the elements within it. The main features extracting methods are divided into two categories: (1) handcrafted extraction and (2) automatic learning. Handcrafted detectors mean that the image operations for computing them (like gradients computation) are chosen "manually". Learning the features implies on the other side to let the model update its weights by computing the classification error, like for neural networks (deep learning). In the next chapter both advantages and drawbacks of the two approaches will be seen.

From classification to detection While it is relatively easy from features to perform classification task (only a yes/no question: "do we have at least one pedestrian in the image?"), detection is quite more complicated since it requires exact localization within the frame and even exact counting of the instances. Different algorithms performing this task will be also presented in the next chapter.

1.2.2 Issues due to deep learning

Small quantity of labeled detection instances Unlike handcrafted pedestrian features extraction methods, learning features by training a large network (that is why it is called *deep* learning) needs a huge quantity of annotated data to prevent overfitting. But we do not necessary have this large amount of data since it should be beforehand manually annotated by humans. Some existing methods suggest tricks to artificially enlarge the available data in the training set or use simply unsupervised learning which by definition does not need labels. Moreover, the more data are rich in variety, the more accurate the detection is. This fact should guide us while building the training set. It reminds us the problem with the background: showing to the network crowded urban environment but empty natural landscapes can create a bias which leads the network to learn how to recognize a city instead of a pedestrian.

Computation time An other drawback of automatic features extraction is the expensive computation time and the higher hardware requirement than for handcrafted features due to the large architecture of the network: lots of layers and wide layers increase the computation time for the back-propagation error. In this context, GPU (Graphics Processing Unit) will be used among others for parallelization of some computations, leading to considerable time optimization.

Concerning the convolutional neural network in particular, compared to standard feedforward neural networks with similarly-sized layers, CNNs have much fewer connections and parameters which are shared thus they are still easier to train.

1.2.3 Structure of this work

Dealing with the previous cited raised issues, we propose a brief review of object and pedestrian detection state of the art in the next chapter (Chapter 2). In Chapter 3, the chosen network architecture for the pedestrian detection task considering the state of the art is described, then evaluated. Finally, in Chapter 4 the obtained results are discussed and this master thesis work is concluded.

Chapter 2

State of the Art

2.1 Object and pedestrian detection

This chapter quickly reviews the state of the art in automatic object detection and localization, with particular attention to human or pedestrian detection. Pedestrian detection is actually a canonical instance of object detection. This is why, pedestrian specific as many as not specic (unspecic) methods used in the object detection task in general are compiled here.

Most of the current work on pedestrian detection have focused on (1) building representative features which should capture the most discriminative information of pedestrians and (2) the detection algorithm that relies on these descriptors. Therefore, this is also our choice to present here the main relevant done work on these two research directions.

Compared to other pedestrian detectors, convolutional networks currently underperformed in comparison with handcrafted features even if they succeed in lots of recent computer vision challenges [2]. But although manual extracted features give a good description for pedestrians, they can not learn the essential characteristics and have bad adaptability. Consequently once again, it is chosen to describe as many of deep learning based features extractors as others methods not involving neural networks.

Nevertheless the basic general pipeline for object detection (figure 2.1) is common for the two approaches:
(1) At learning time, an object detector is constructed from a large number of training examples.
(2) At test time, the detector is then "scanned" over the entire input image at different scales and positions in order to find a pattern of intensities which can figure the presence of the target object.
The next two parts are first for object features extraction and secondly for the way to use classification task for the detection.

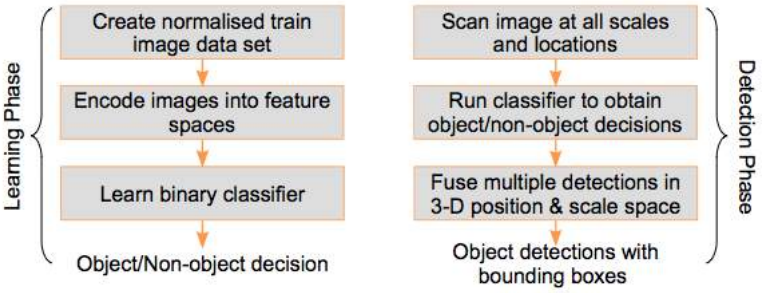

FIGURE 2.1: Object detection task pipeline
image from [3]

2.1.1 Features learning

The main feature extracting methods are divided into two categories: handcrafted extraction and automatic learning.

2.1.1.1 Four handcrafted features extraction methods

The progress of the last decade for the visual recognition has been founded essentially on handcrafted (1) Haar-like [4, 5], (2) SIFT [6] and (3) HOG [3] features. This features are designed to be insensitive to intra-class variation but dependant on inter-class variation. Haar-like, SIFT and HOG features enable to dig out the global shape of pedestrians. In particular, SIFT and HOG are blockwise orientation histograms which extract gradient structure (= edges) that is typical of local shape. A last handcrafted features extraction method is also briefly described : (4) Integral Channels Features [8].

Haar-like features [4, 5] This filter is based on appearance and motion information (extracted from two consecutive frames) by measuring the differences between region averages at various scales and orientations. For instance, regions where the sum of the absolute values of the differences is large point motion, while the difference between shifted versions of the next contiguous frame with the considered image reveals the direction of the motion. AdaBoost [7] is then used for training a cascade of more and more sophisticated classifiers for non-object region rejection.

SIFT features The Scale Invariant Feature Transform (SIFT) method [6] enables to compute for an image distinctive image scale, orientation, viewpoint, translation and illumination invariant features. A difference-of-Gaussian function identifies first potential interest points which are invariant to scale and orientation. At each candidate location, a model is fitted to determine location and scale and key points are selected thanks to their stability through consecutive frames. Then, one or more orientations are associated to each key point location according to local image gradient directions. Finally, to describe the key point, the local image gradients are measured at the selected scale in the region around each key point which gives after transformation a robust representation for some shape distortion and change in illumination.

HOG features Histograms of Oriented Gradients (HOG) features [3] have been specially designed for human detection. The method roughly evaluates normalized local histograms of image gradient orientations in the dense grid formed by the image divided into small *cells*. For each cell a local 1-D histogram of gradient directions (= edge orientations) is built over the cell pixels and the set of these histograms gives the representation.

Integral Channel Features A *channel* of an input image is a registered map where the pixels are computed from corresponding patches of input pixels, preserving global image layout. For a grayscale image, the trivial channel is simply the input image itself. For a color image, the red, green, blue (RGB) levels are 3 possible channels forming three matrices of values between 0 and 255 corresponding to each pixel. In [8], numerous registered image channels are computed using linear and non-linear transformations of the input image. Local sums, histograms, Haar features or their variation applied to those channels constitute then the final features.

Have been reminded here the most important handcrafted features extractors but others exist like part-based model methods which reflect human articulations and postures. It has been chosen not to develop them because it is out of the scope of this limited review.

Despite the effectiveness of those detectors, it can be noticed that during the period 2010-2012, the progress has been quite negligible and obtained by combinations and/or by using minor variants of this previous cited methods. Can deep learning inverse the trend?

2.1.1.2 Automatic features learning

Visual recognition in the primate brain visual cortex is enabled by different areas interconnected in a multi-stage hierarchy. Neural networks computer vision is inspired by this biological phenomenon which suggests indeed that there might be hierarchical multi-stage processes for computing features that are even more informative than those obtained with SIFT or HOG methods for visual recognition.

In 1980, Fukushima [9] first proposed the basic "neocognitron" which is a biologically-inspired hierarchical and shift-invariant model for pattern recognition. But the concept of supervised training came after: in 1986, Rumelhart *et al.* [10] introduced internal representations learning by error propagation and in 1989, LeCun *et al.* [11] finally extended the *neocognitron* model class by effectively training convolutional networks (CNNs) in a supervised fashion with error back-propagation. After that, in the late 80's and early 90's, his algorithm, ConvNet, has been used for automatic reading of amounts on bank checks, automatic reading of zip code on mailing envelopes and for document recognition in general [12].

The rise of Support Vector Machines (SVM) and the prohibitive computing time for training deep neural network architectures caused the CNNs almost to be dropped after. But in 2006, Hinton *et al.* [13] found a fast way to train such networks through results on unsupervised training/unsupersived pre-training followed

by supervised fine-tunning. This revived the interest for representation learning as opposed to classifier learning. Then, in 2012, Krizhevsky *et al.* [14] achieved a breakthrough by reaching considerably higher image classification accuracy on the 1000-class ImageNet Large Scale Visual Recognition Competition (ILSVRC) [15, 16].

Finally, in the domain of automatic features learning, it is also relevant to point three other works :
(1) the ConvNet structure twisted by Sermanet *et al.* [17] which uses the original pixel values as the input and uses both unsupervised and supervised methods to train multi-stage automatic sparse convolution encoder.
(2) UDN (Unified Deep Net) [18], a joint deep neural network model combined with a deformation and occlusion model.
(3) Mixes of manual features extraction methods with deep learning models [19].
But now how to go from image classification to object detection?

2.1.2 From image classification to object detection

At test time, a trainable classifier such as SVM [3, 20], boosted classifiers [5, 8] or random forests decides whether a candidate window (a sub-frame of the image) shall be detected as enclosing a pedestrian. But in the case of neural networks, the last fully connected layers can easily play the role.

Then, to go from image classification to object detection, that means, to solve the localization problem, the main popular methods are based on a *sliding window approach*. At test time, the learned pedestrian/object detector will be scanned across the input image at multiple scale and locations.

Typically, the first step is to create at multiple scales and locations *detection window candidates* from the original image while the second is to apply the classifier on this sub-frames. The last step is to correctly merge the different results on the windows. The three following parts focus on this.

2.1.2.1 Generate candidate windows

By generating multiple candidate sub-frames at multiple scales and locations (the process is called multi-scaling) it is easily infered that the created sub-frames are potentially of very different sizes. How will the detector deal with this? Should the detector have a fixed dimension?

Multi-scaling approaches For dealing with the multi-scaling, there are two approaches.

(1) In [4], the *detector* is scaled itself. This process is justified by the authors Viola and Jones by the fact that the features can be evaluated at any scale with the same computation cost. Their detector is also scanned across location. Subsequent locations are obtained by shifting the window of δ pixels (in practice $\delta = 1.0$).

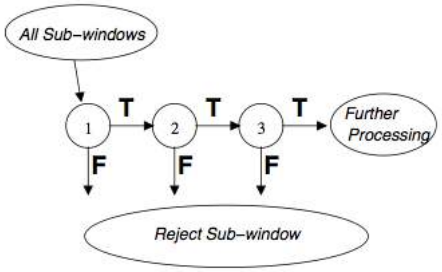

FIGURE 2.2: Reducing the number of sub-window candidates for detection scheme
image from [4]

The shifting process depends on the scale of the detector: if the current scale is s, the window is shifted by $round(s \times \delta)$. The choice of δ affects the speed of the detector as well as its accuracy.

(2) In [20], this is the *candidate sub-windows* (remember, not likely of the same size) which are scaled so that they fit the dimensions of the detector.

As the binary classifier based object detector sweeps a detection window across the image at multiple scales and locations, we can wonder whether some computation time could not get saved by rejecting rapidly some obvious negative candidate sub-windows.

Reducing the number of candidate sub-windows [4] attacks the problem of reducing the number of candidate sub-windows. Their detection schema is as in figure 2.2. A succession of classifiers are applied to actually every sub-frames of the input image but the initial classifier already exludes many negative instances and this, with little processing. Again, subsequent layers eliminate more negatives but are more time consumming. After several stages in the classifiers model the number of sub-frames have been quite good reduced.

In [20], the approach is quite different. The model does not eliminate rapidly negative windows but make a selection at the root. *Regions with CNN features (R-CNN)* algorithm generates only 2000 category-independent *region proposals* (sub-frames) from the input image with the *selective search* principle which consists in preselection of potentially good locations for classification thanks to segmentation of the whole image [21]. Then it extracts a fixed-length feature vector from each proposal using a CNN, and finally classifies each region with category-specific linear SVMs. The region proposals do not have the same shape but the CNN needs a fixed-size input, so the sub-image from the region proposal is warped.

2.1.2.2 Integration of multiple detections

Since the final pedestrian detector is supposed to be invariant to small changes in translation and scale, multiple detections will naturally be found around each same pedestrian in a sweeped image. To merge

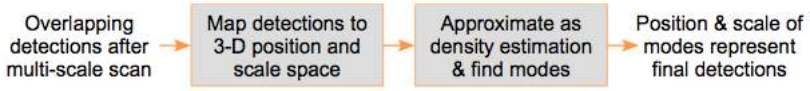

FIGURE 2.3: Detection part basic pipeline
image from [3]

those overlapping detections into a single one, two approaches are mainly used: the *disjoint subsets* and the *non-maximum suppression* algorithms.

Disjoint subsets In [4] the set of detections are first split into disjoint subsets. The rule is : "Two detections are in the same subset if their bounding boxes overlap". Then, each partition produces a single final detection and the corners of the final bounding region are the mean of the corners of all detections in the set.

Non-maximum suppression In [3], the subtlety is that windows used to learn binary classifiers can be larger than the object to allow some context. They have 16-pixel margin on each side of a person. Thus some true distinct detections may be overlapping and classified as a single detection by previously described Viola and Jones method [4]. A typical example is when the image contains two people occurring at different scales which yields to one detection (one occurring in another detection). So Dalal introduced a new method for merging the multiple detections: the *non-maximum suppression* which is based on representing detections in a position scale pyramid (see figures 2.3 and 2.4). Each detection provides a weighted point in this 3-D space (height, width, scale) and the weights are the detection's confidence score. The corresponding density function is estimated and from the value peaks we find the final detections, with positions, scales and detection scores.

Finally an other twist: In [22], which applies non-maximum suppression too, each bounding box detection is scored, the highest scoring ones are kept and the bounding boxes that are at least covered by 50% by an other previously selected are dismissed.

But now, can detection results be obtained by basic methods improved with some tricks?

2.1.2.3 Improving results

At least two tricks exist for improving the localization results given either by the disjoint subsets or the non-maximum suppression algorithm.

In [4], a simple voting scheme is applied at the test time to further improve results. Running for instance three learned detectors (the most possible independent of course) and outputting the majority vote of the three detectors improve the detection rate and eliminate more false positives.

In [20], a bounding box regression is used for reducing localization errors.

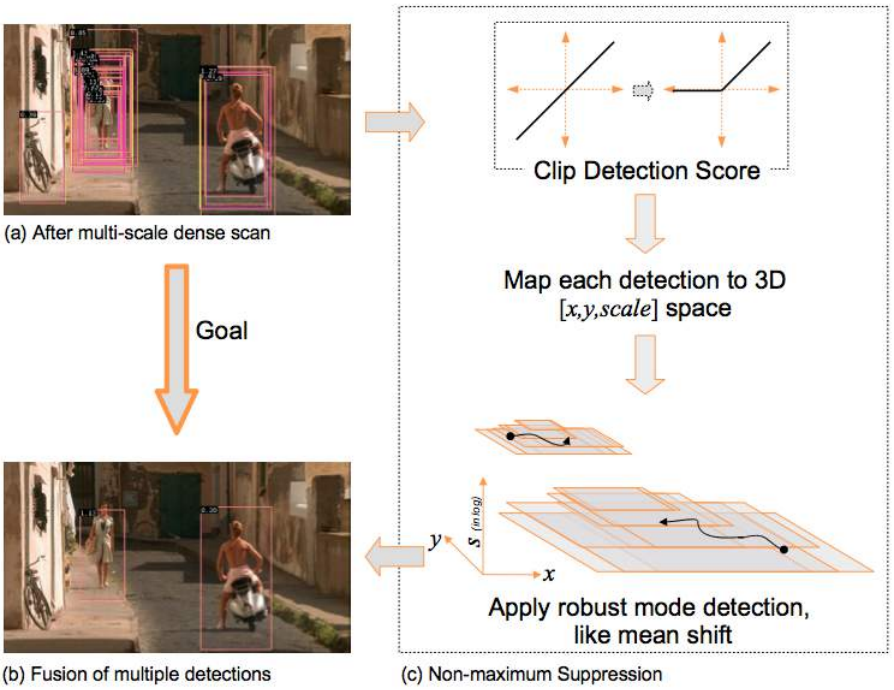

FIGURE 2.4: Multi-scaling and merging results
image from [3]

The inherent difficulties of object/pedestrian detection have been explained above. The next part briefly focuses on the typical deep learning problem of small labeled datasets.

2.1.3 Dealing with scarce labeled data

A large CNN (in height, in width) needs lots of labeled data for the supervised training. In case of data lack, papers performing the object or pedestrian detection task offer some tricks. For instance, CNNs are first *unsupervised* pre-trained (with auto-encoders) and then *supervised* fine-tuned in [17]. [20] showed even that pre-training on a large dataset not specific to the concerned task followed by domain-specific fine-tuning on a small dataset is effective.

This aspect of deep learning will be more developed in the consecrated next part with the problem of over-fitting.

2.2 Convolutional neural networks

In 2012, Krizhevsky *et al.* [14] reached considerably higher image classification accuracy on the ImageNet Large Scale Visual Recognition Competition (ILSVRC) [15, 16] by training a large CNN on 1.2 million labeled images. The network contained eight learned layers: five convolutional and three fully-connected. This breakthrough marks a revival for deep learning in computer vision.

In this part, what is exactly a convolutional network will be seen as well as the different "tricks" for computation optimization and avoiding overfitting.

2.2.1 Definition and general concepts

2.2.1.1 General overview

One or more convolutional and subsampling layers, followed by one or more fully connected layers (as in a standard multi-layer neural network) form what we call a *Convolutional Neural Network* (CNN). The initially random weights of the CNN are iteratively trained to minimize the classification error on a set of labeled training images. Generalization performance is then validated on a different set of test images.

2.2.1.2 The role of convolution kernels

The convolutional layers contain convolutional kernels. A *(convolution) kernel*, also called *convolution matrix, filter, receptive field, mash* is a small matrix useful for image processing since it enables to blur, detect edges, shapes, etc. from objects in the image. So relevant information from the image channels can be filtered by applying the convolution matrix on every small portions (patches) of the input image.

The output of the kernel is the modified image called a *feature map* with one for every color channel. The value of each pixel of the output image are computed by multiplying each kernel value by the value of the corresponding input image pixel. (See Appendix A for more explanations.)

Generating manually this feature map is tedious since there is one feature map for on each type of problem. So instead of having fixed weights in the kernel, parameters are assigned to this and are learned by the convolutional network.

Then not a single kernel is learned by convolutional nets, but a hierarchy of multiple ones at the same time, corresponding to different description abstraction level. These features corresponding to the learned map then provide the inputs for the next kernel which filters the inputs again.

Once hierarchical features are learned, they are simply passed to a fully connected network that classifies the input image into classes, for example.

Each convolutional layer have different parameters :
- the number of expected features map from the previous layer $nInputPlane$
- the number of output features map the convolution layer will produce $nOutputPlane$
- the kernel width of the convolution kW, default is 1
- the kernel height of the convolution kH, default is 1
- the step of the convolution in the width dimension dW

- the step of the convolution in the height dimension dH
- the additional zeros added per width to the input planes (features maps) $padW$. Default is 0, a good number is $\frac{kW-1}{2}$
- the additional zeros added per height $padH$ to the input planes. Default is $padW$, a good number is $\frac{kW-1}{2}$

If the input image is a 3-D tensor of dimension $nInputPlane \times height \times width$, the output image size will be $nOutputPlane \times oheight \times owidth$ where:

$$owidth = floor(\frac{width+2*padW-kW}{dW} + 1)$$
$$oheight = floor(\frac{height+2*padH-kH}{dH} + 1)$$

In case of a pooling layer, the output has the input's spatial dimensions divided by the size of the pooling kernel.

So convolution layers at each stage reduce the spatial dimensions of features. If we want to keep the spatial dimension, we can use previously cited *padding* which consists in adding pixels (basically blank pixels) around the features map. Actually, padding is also used to avoid borders effects: The pixels which were previously at the borders of the applied filter are now in the middle of the same filter, thanks to adding blank pixels around them.

2.2.1.3 The role of fully connected layers

The fully connected layers act as classifier. One fully connected layer plays the role of a linear classifier while from two layers, the classifier becomes non linear. The nonlinearity enables to learn more complicated patterns but too much layers can lead to overfitting on the data.

2.2.1.4 The role of pooling layers

Pooling is a way of sub-sampling, i.e. a way for reducing the dimension of the layer input. For each group of a fixed number of layer units it yields a single value. For *max-pooling*, for example, this single value is the maximum. In that way, the pooling layers action in CNNs can be interpreted as the sum up of the outputs of the neighboring group of neurons in the same kernel map. A pooling layer can be seen as a kind of grid of pooling units spaced s pixels apart, each summarising a neighborhood of size $z \times z$ centred at the location of the pooling unit. The commonly employed local pooling is for $s = z$. If $s < z$, (this means the stride between two groups is smaller than the number of units in the group), we talk about overlapping pooling. In [14], $s = 2$ and $z = 3$, which improves results in comparison with $s = z = 2$ (output of equivalent dimensions). It is generally observed during training that models with overlapping pooling are less concerned by overfitting.

Max-pooling layers follow both response-normalization layers as well as the fifth convolutional layer in [14]. In [23], Scherer *et al.* found that max-pooling can lead to faster convergence, select better invariant features, and improve generalization.

2.2.2 Properties of activation functions

2.2.2.1 ReLU Nonlinearity

The standard fashion to model a neuron's output f (activation function) as a function of its input x is $f(x) = tanh(x)$ or $f(x) = (1 + e^{-x})^{-1}$. In terms of training time with gradient descent, these saturating nonlinearities are much slower than the non-saturating nonlinearity $f(x) = max(0, x)$ proposed by Nair and Hinton in [24] where neurons with this nonlinearity are called *Rectified Linear Units (ReLUs)*. Deep convolutional neural networks with ReLUs train several times faster than their equivalents with *tanh* units, so it is applied to the output of every convolutional and fully-connected layers in [14].

The ReLUs' benefit is that they do not need input normalization to avoid saturating. If at least some training examples produce a positive input to a ReLU, learning will happen in that neuron. However [14] still find that local normalization helps generalization. Therefore, response-normalization layers follow its first and second convolutional layers.

Very recently [25] proposed a generalization of ReLU which improves the state-of-the-art accuracy on ImageNet 2012 classification dataset.

2.2.2.2 PReLU Nonlinearity

PReLU (2015) stands for Parametric Rectified Linear Unit and is a generalization of ReLU. Formally, this activation function is defined as:

$$f(y_i) = \begin{cases} y_i & if \quad y_i > 0 \\ a_i y_i & if \quad y_i \leq 0 \end{cases}$$

with y_i the input of the nonlinear activation f on the ith channel and a_i is a coefficient controlling the slope of the negative part. The index i indicates that the nonlinear activation varies on different channels. Notice that when $a_i = 0$, it is simply ReLU. PReLU introduces a new number of parameters equal to the total number of channels, which is negligible in comparison with the total number of weights, so no more risk of overfitting is expected.

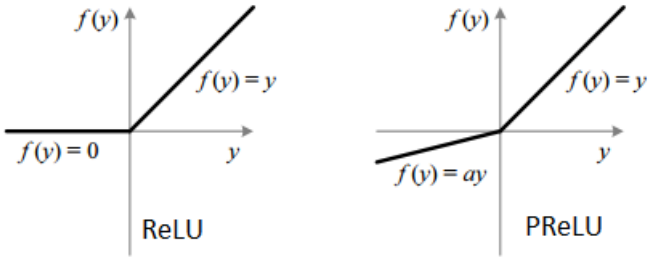

FIGURE 2.5: ReLU vs PReLU
image from [3]

2.2.2.3 Maxout model

In [26] (2013), there is an other type of activation function: the *maxout* unit. Given an input $x \in \mathbb{R}^d$, a maxout hidden layer implements the function $h_i(x) = max_{\ j\in[1,k]}\ z_{ij}$ where $z_{ij} = x^T W_{...ij} + b_{ij}$ and $W \in \mathbb{R}^{d\times m\times k}$ and $b \in \mathbb{R}^{m\times k}$ are learned parameters.

General bricks for a CNN architecture have been seen. Now additional tricks will be shown which enables to fight overfitting, the peeve of machine learning.

2.2.3 Preventing overfitting

2.2.3.1 Data augmentation

A first simple mean to reduce overfitting on image data is to artificially extend the number of available data in the dataset thanks to transformations which alter sufficiently the image to give the impression to the network to dispose of a new one, but a reasonable transformation though, so that the object is still recognizable [14, 27–29]. Such an operation is called *data augmentation*. Two kinds of data augmentation can be distinguished:
(1) to generate image translations, horizontal reflections or other elastic deformations.

(2) Second one: to modify the RGB channels values in training images. To each training image, multiples of the found principal components after application of PCA (Principal Component Analysis) on the set of RGB pixel values, are added with magnitudes proportional to the corresponding eigenvalues times a random variable (drawn from a Gaussian with mean zero and standard deviation 0.1). Therefore to each RGB image pixel $I_{xy} = (I^R_{xy}, I^G_{xy}, I^B_{xy})^T$ the following quantity is added: $(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)(\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3)^T$ where $\mathbf{p}_i$ and λ_i are the ith eigenvector and eigenvalue of the 3×3 covariance matrix of RGB pixel values, respectively, and α_i is the above mentioned random variable. Each α_i is drawn only one for all pixels of each new submitted training image. This scheme enables to take in account the fact that object identity is invariant to changes in the intensity and color of illumination.

2.2.3.2 Dropout

In machine learning, a conjunction of many different models is a good mean to decrease test errors, but to avoid too expensive resulting computation, the "dropout" technique has been introduced in [30]. It consists in resetting the output of each hidden neuron with probability 0.5. The neurons which are "dropped out" then do not contribute to the forward pass and do not participate in back-propagation. This technique make complex co-adaptations of neurons smaller, since a neuron can not rely on the presence of particular other neurons. In [14], a little variant is applied: At test time, all the neurons are used but their outputs are multiplied by 0.5.

In this chapter focusing on a brief state of the art on pedestrian detection and deep learning, have been studied how to extract manually and automatically features from images, how to perform object detection from classification task, how work the CNNs and some tricks for computation optimization and for preventing overfitting. Now in the next chapter, our propositions and their evaluation will be presented.

Chapter 3

Propositions, evaluation and results

First, before detailing our propositions for the pedestrian detection task, the general pipeline for our work is exposed. There are two main steps in the global project within my internship is incorporated: (1) pedestrian classification and (2) pedestrian detection. For the moment, progress is currently made on the pedestrian classification task.

Pipeline:
(1) Write a data sampling library including data augmentation.
(2) Train a CNN-based classifier for pedestrians and test it on carefully crafted test set.
(3) Compare against state-of-the-art handcrafted descriptors from the literature.
(4) Detection task

The work implementation has been made in Lua language in the Torch framework.

3.1 Building a relatively large dataset

3.1.1 How to build a good dataset

A first important step in this work is to build a (as large as possible) relevant dataset for the pedestrian detection challenge. The most possible data variety is wished to prevent overfitting. For completing this task, has been decided to merge different datasets which are (1) specific for pedestrian detection but also which are (2) non-specific and designed for object detection, still containing persons though. (Please be careful, a person is quite different from a pedestrian because it can be sitting somewhere else rather than walking down the street.) The choice of existing datasets should be guided by the following questions :
- Do we have a single pedestrian or are the frames crowded?
- How many different pedestrians do we have in the dataset?
- Do the frames come from photos or videos?

FIGURE 3.1: Daimler dataset sample

FIGURE 3.2: ETH dataset sample

- Do we have lots of negative frames?
- Does the lighting vary?
- Do we have a variety of different points of view? (from surveillance videocamera, from a car, from people...)
- What is the quality, the size of the frames? Are these black and white frames or color ones?
- What is the average size of a pedestrian in the dataset?
- Are the pedestrians occluded or not?
- Do we have a variety of backgrounds?
- Are the frames annoted? How?

These following benchmarks databases have been eventually selected to be merge : Daimler [31], ETH [32], INRIA [33], TudBrussels [34], Caltech-USA [35], MSCOCO [36], PETA [37], CBCL [38] , CVC [39] datasets. They are briefly described and four instances per dataset are shown.

3.1.2 Description of the merged datasets

Daimler dataset Daimler consists of black and white videos (the only ones) captured from a car going through german streets: figure 3.1.

ETH dataset It is a mid-sized video dataset and has higher density and larger scale than the remaining datasets: figure 3.2.

INRIA dataset INRIA dataset has high quality annotations of pedestrian in diversity settings from holiday photos (city, beach, mountains, etc.), which is why it is commonly selected for training: figure 3.3.

TudBrussels dataset It is a mid-sized video dataset in an urban environment: figure 3.4.

Caltech-USA dataset The Caltech dataset is one of the most popular pedestrian detection datasets. Again it is videos rolled from a car going through US streets under good weather conditions. The first six videos

FIGURE 3.3: INRIA dataset sample

FIGURE 3.4: TudBrussels dataset sample

FIGURE 3.5: Caltech-USA dataset sample

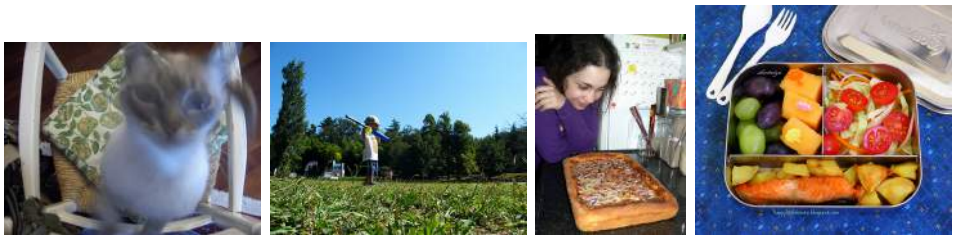

FIGURE 3.6: MSCOCO dataset sample

are often used for training and the five remaining videos for the validation: figure 3.5. In the dataset, every frames with stride 30 is kept.

MSCOCO dataset The MicroSoft Common Objects in Context (2014 release) is a dataset for object detection in general with photos containing 80 objects types: figure 3.6.

PETA dataset The PETA dataset consists of 19000 already cropped pedestrians with lots of diversity in lighting, point of view etc.: figure 3.7.

CBCL dataset It is a collection of street scenes taken from the sidewalk, cross walk or the street, in good weather condition during the day: figure 3.8.

CVC dataset This is a partially occluded pedestrian dataset. The dataset consists of 593 positive frames with annotated partially occluded pedestrians: figure 3.9.

FIGURE 3.7: PETA dataset sample

FIGURE 3.8: CBCL dataset sample

FIGURE 3.9: CVC dataset sample

3.1.3 Available annotations

The datasets give the annotations in different formats: JSON files (MSCOCO) or Caltech format text files from [35] (Daimler, ETH, INRIA, TudBrussels, Caltech-USA, CVC) or other text format (CBCL), mostly indicating the class of the annotated object, the coordinates of the upper left corner x and y (with origin on the upper left corner of the original frame), the width w and the height h of the crop for a rectangular crop, but sometimes all the points surrounding the person (CBCL). What is called later a *bounding box* is the rectangle enclosing the pedestrian and this is defined by the four elements: x, y, w, h. The following classes are available in the cited datasets:

- Person: The main label for positive annotations, the pedestrian is easy to detect. The following examples are from respectively: INRIA, CVC, TudBrussels and ETH (the annotations are red rectangles), figure 3.12.

- Ignore: Ignore label regroups annotations from Daimler dataset previously forgotten or voluntary ignored by the first annotations. The ignore label refers to partially or totally occluded pedestrian. Examples from the Daimler dataset (the annotations are red rectangles): figure 3.11.

- People: This label is for a group of people which is hard to split into different distinct pedestrian crops. Examples from the dataset Caltech-USA (the annotations are red rectangles): figure 3.12.

- Person? : This label is assigned when obvious identification of a pedestrian is very hard. Examples from the dataset Caltech-USA (the annotations are red rectangles): figure 3.13.

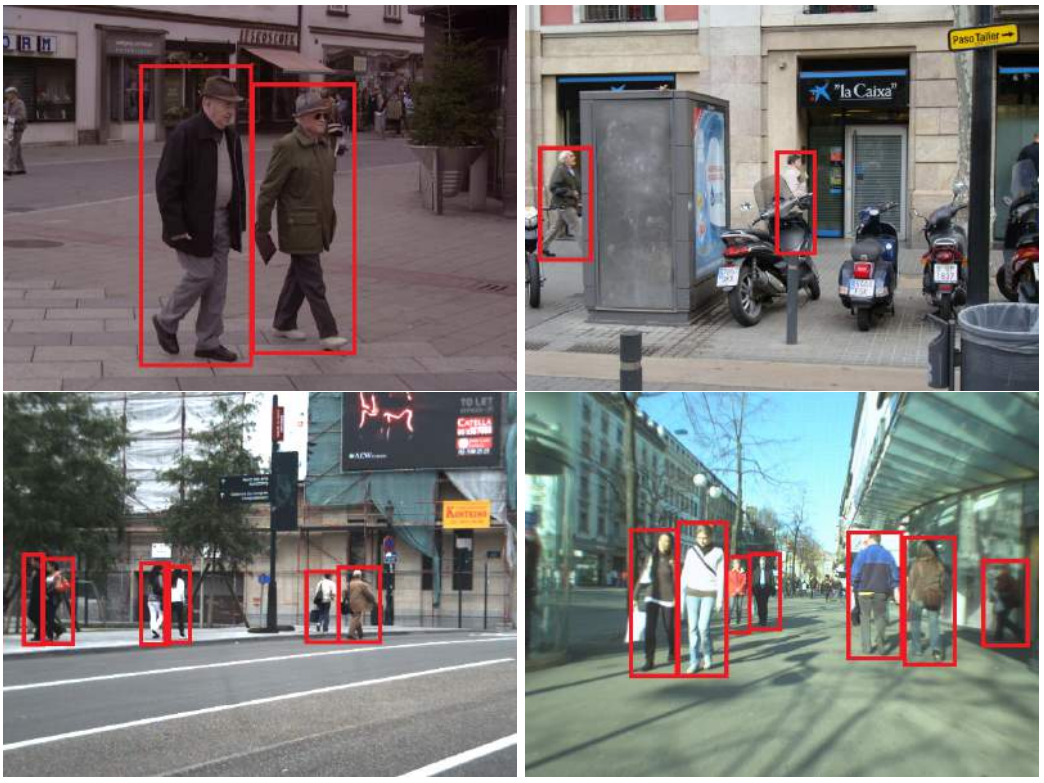

FIGURE 3.10: person class annotations from INRIA, CVC, TudBrussels and ETH datasets sample

FIGURE 3.11: ignore class annotations from Daimler dataset sample

FIGURE 3.12: ignore class annotations from Caltech-USA dataset sample

FIGURE 3.13: person? class annotations from Caltech-USA dataset sample
We can see that the pedestrian of this class are difficult to identify.

FIGURE 3.14: person-fa class annotations from Caltech-USA dataset sample

- Person-fa : This label is quite the same as person?. The difference between the two labels is unclear but what is sure for both labels is that the corresponding pedestrians in these crops are hard to recognize, because of shadow, brightness, occlusion or smallness. Examples from the dataset Caltech-USA (the annotations are red rectangles): figure 3.14.

As the given annotations are not always of good quality, it has been decided that respect of some conditions is required before an annotation could be added to the dataset:

- the dimensions of the annotation w and h should be strictly positive
- $x + w > 5$
- $y + h > 5$

Actually, some annotations begin outside the frame itself. For the statistics of our final dataset, see tables 3.1, 3.2 and 3.3.

That is all for the positive crops feeding our positive set for the training set. During training time, negative crops have also to be shown to the net after been extracted/sampled from all scales in the frames of the dataset (frames already containing positive crops or not) in order to avoid a bias on the zoom. The dimensions of a negative crops are related to the size of a mean pedestrian.

N	dataset	source	format	colour	size	nb frames	nb annots
1	Daimler	video	PNG	no	640x480	21790	56484
2	ETH	video	PNG	RGB	640x480	1804	14166
3	INRIA	holidays photos	PNG	RGB	varies	2120	1826
4	TudBrussels	video	PNG	RGB	640x480	508	1498
5	USA	video (ev. 30 frames)	PNG	RGB	640x480	8274	11504
6	MSCOCO	photos	JPG	RGB	≈ 578x484	123287	269886
7	PETA	crops	varies	RGB	≈ 72x170	19000	19000
8	CBCL	closed photos	JPG	RGB	1280x960	3547	1449
9	CVC	photos	PNG	RGB	640x480	593	2008
					Total	**57636**	**377821**

TABLE 3.1: Basic information on datasets

N	dataset	av. per frame	Person	Ignore	People	Person?	Person-fa	**Total**
1	Daimler	2.59	14131	40925	1428	0	0	**56484**
2	ETH	7.85	14166	0	0	0	0	**14166**
3	INRIA	0.86	1826	0	0	0	0	**1826**
4	TudBrussels	2.95	1498	0	0	0	0	**1498**
5	USA	1.39	9479	0	1636	258	131	**11504**
	USA train	-	*5080*	*0*	*1152*	*92*	*41*	*6365*
	USA test	-	*4399*	*0*	*484*	*166*	*90*	*5139*
6	MSCOCO	2.18	269886	0	0	0	0	**269886**
7	PETA	1	19000	0	0	0	0	**0**
8	CBCL	0.40	1449	0	0	0	0	**1449**
9	CVC	3.39	2008	0	0	0	0	**2008**
		Total	**333443**	**40925**	**3064**	**258**	**131**	**358821**

TABLE 3.2: Information on classes

N	dataset	mean w	mean h	std w	std h	mean w/h
1	Daimler	27	54	20	40	0.51
2	ETH	51	101	32	63	0.50
3	INRIA	119	289	61	148	0.41
4	TudBrussels	28	74	14	37	0.39
5	USA	24	59	19	46	0.43
6	MSCOCO	82	133	98	131	0.64
7	PETA	72	170	22	55	0.43
8	CBCL	89	195	67	117	0.46
9	CVC	54	142	29	70	0.38

TABLE 3.3: Information on person crops dimensions (in pixels)

3.2 Sampling algorithm

The first assumed sampling algorithm is algorithm 1. Notice that $rand\_*(parameters)$ are functions for choosing randomly an object * with respect to the parameters. $get\_bb(annot)$ establishes the bounding box from the positive annotation information. $getBB\_in\_frame(frame)$ gives the list of bounding boxes in the current frame *bb_list* which is very useful when a negative crop is supposed to be built (with $create\_bb(frame, bb\_list)$).

Roughly, it first decides whether a positive or a negative crop is drawn for the network. Then a dataset is chosen where can be drawn this annotation. If a positive crop is wanted, it is just picked from the database

Algorithm 1 Sampling algorithm

```
isPositive ← 0 or 1
dataset ← rand_dataset(isPositive)
if isPositive == 1 then
    class ← rand_class(dataset)
    annot ← rand_annot(class)
    while the constraints in parameters are not respected do
        annot ← rand_annot(class)
    end while
    bb ← get_bb(annot)
else
    while we can't build a neg bb respecting the constraints in parameters do
        frame ← rand_frame()
        if frame contains no positive annotations then
            bb ← create_bb(frame)
        else
            bb_list ← getBB_in_frame(frame)
            bb ← create_bb(frame, bb_list)
        end if
    end while
end if
return bb
```

by choosing a wanted positive class (person, ignore ...). Maybe some constraints have to be respected which will be explained below. If a negative crop is wanted, it has to be built among either the negative frames ("empty" frames) or positive frames, that means they already contain positive annotations which should not (or at least not so much) be overlapped. A negative crop should respect constraints too, for instance it could be relevant that the dimensions follow those of a positive crop. See for this, the statistics on the mean and the standard deviation of pedestrian crops for each dataset in table 3.3.

The method will be now explained in detail but first let us take a look at the numerous available choice parameters of the sampling algorithm.

3.2.1 Multiple choice parameters

The algorithm is guided by multiple parameters which should be properly indicated on a parsed configuration file:

- the probability *datasets_choice_train* to choose each dataset during training time ...

- ... and *datasets_choice_val* during test time. It enables to find the right combination for maximizing the results. Indeed, it has to be dealt with very different datasets, one in black and white, others with no negative frames etc. The sum of each probability has to be equal to 1.

- at each draw of bounding box, a probability *pos_proba* to choose a positive bounding box (= containing a pedestrian). A good value is 0.5.

- among the positive annotations, there are different classes as cited above. It can be chosen for each dataset which class should be considered among the positive annotations. The parameters are $ignore\_bool\_table$, $people\_bool\_table$, $personQP\_bool\_table$, $person\_fa\_bool\_table$. If a particular class is authorized, annotations from this class will be present in training and test sets with the corresponding proportion. See table 3.3 for figures.

- the minimum width w_bb_min and height h_bb_min for the extracted bounding box (in pixels). A dimension below 5 pixels is hardly exploitable. In our case, it is particularly relevant to have rectangular crops with a height larger than the width to keep the proportion of a pedestrian.

- only for positive annotations, it can be decided how far the bounding box is allowed to be from the borders of the frame, in percentage of the dimensions. $allowed\_dist\_from\_bounds = 0$ indicates that the bounding box can be directly adjacent to the borders of the frame; a positive value requires that the bounding box is not adjacent to the borders while a negative one allows that a bounding box overflows outside the frames: in any case, this means the whole pedestrian will not be in the final selected crop.

- two methods for generating negative annotations height and width are available $method\_rand\_hw = 1$ *or* 2

- for generating negative bounding boxes, a ratio for the maximum authorized overlap p between 0 and 1 can be fixed, it can be thought in terms of Jaccard index too: $jaccard\_index$.

- can be fixed a maximum proportion for the width $ratio\_w\_bb\_max$ and the height $ratio\_h\_bb\_max$ of the bounding box in comparison with those of the frame

- the probability to take the left-right reflection during test time is up to the user

- the probability to draw hard negative annotations (= negative crops which have been previously wrong classified) too when a negative one is needed.

The game is then to find optimal parameters allowing to have the best well-classified frames rate for the positive and negative class. Now will be explained a little bit more (1) our two methods for generating size of negative bounding boxes, (2) given height and width two methods for finding a negative bounding box with overlap constraints respect.

3.2.2 Two detailed methods for negative crops: generating height and width

Method 1: Dependance of two gaussian distributions In this method, h follows a gaussian distribution with parameters μ_h and σ_h^2, the mean and the variance respectively of the height of a pedestrian in the considered dataset. Similarly, w follows $\mathcal{N}(\mu_w, \sigma_w^2)$. So w knowing h follows $\mathcal{N}(\mu_w + \frac{\sigma_{hw}}{\sigma_w}(h - \mu_h), \sigma_w - \frac{\sigma_{hw}^2}{\sigma_h})$.

Method 2: Independence of the variable h and r $r = \frac{w}{h}$ and r follows $\mathcal{U}(0, 1)$ while h follows $\mathcal{N}(\mu_h, \sigma_h^2)$. In both cases, overlap between an established positive bounding box and a negative one can be allowed. The

following condition with parameter p varying between 0 (no overlap authorized) and 1 is defined: the current considered bounding box is denoted A. It has different overlapping areas of surface c_i with bounding boxes B_i and we want: $max\ _i\ (\frac{c_i}{min\ (A,\ B_i)}) < p$ which is called later "overlap ratio".

3.2.3 Two detailed methods for negative crops: generating a location in a positive frame

Method 1 These last methods to find h and w lead us to a first very simple sampling algorithm for negative crops: With previously determined h and w, it is tried at most 10 times to find a random place in the frame and if it does not fit (the found bounding box is always overlapping a positive one with overlap ratio above p), an other frame is simply chosen. See algorithm 2.

Algorithm 2 Negative crops from positive images sampling algorithm 1

$h \leftarrow generate\_h(dataset, frame)$
$w \leftarrow generate\_w(dataset, frame, h)$
$count = 0$
while $overlap\ \ ratio \geq p$ `and` $count \leq 10$ **do**
    $x, y \leftarrow rand\_coordinates(frame)$
    $count \leftarrow count + 1$
end while
if $count \leq 10$ **then**
    return x, y, w, h
else
    $frame \leftarrow new\_frame(dataset)$
    go to the beginning
end if

This method is a little bit "lazy", though. There is a bias since sometimes a positive frame can be discarded without being sure that a bounding box could fit the available space.

Method 2 This method (algorithm 3) generates h and w like before but determines more surely if there is enough space to welcome the $w \times h$ bounding box, by finding if possible, an area which can welcome the bounding box. Only it is too hard to find one, the frame is discarded.

Notice that it is not a good thing to change the size of h and w instead to fit the frame because an obvious bias is introduced: The algorithm could become lazy and find only tiny bounding boxes which are absolutely not representative of the size of pedestrian who can be met.

This new recursive algorithm relies on the fact that the intersection of two rectangles is still a rectangle. At each met bounding box in the frame, the four biggest rectangles around are determined, as illustrated in figure 3.15. Overlap is controlled by the Jaccard index $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$ where A is the current studied bounding box and B an other existing one in the considered frame. $J(A, B)$ should not be taller than a certain parameter called $jaccard\_index$ between 0 and 1 .

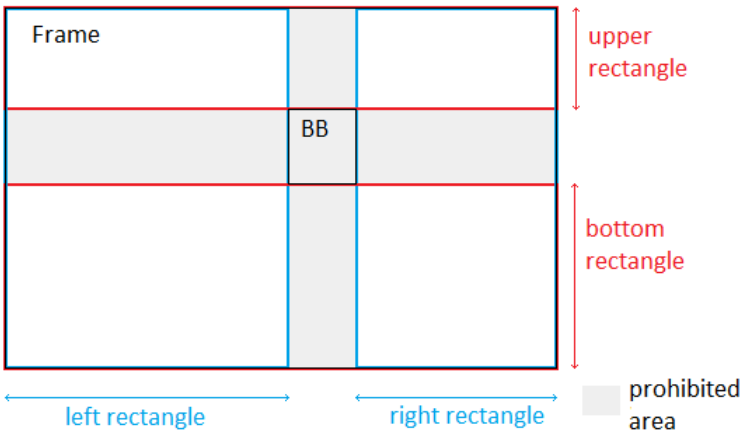

FIGURE 3.15: the four biggest rectangles around a bounding box

The algorithm has for inputs the coordinates of the current considered rectangle x_{env}, y_{env}, w_{env}, h_{env}, the list of bounding boxes in the frame *bbList* and of course the size of the wanted bounding box to be cropped w, h. At the initial state, x_{env}, y_{env}, w_{env}, h_{env} are equal respectively to 0, 0, *image_w*, and *image_h*. The output is the first found rectangle where the bounding box can fit. To ensure that it is not always the same rectangle which is chosen, the processing order of the four rectangles is always randomized.

Algorithm 3 Negative crops from positive images sampling algorithm 2:

$find\_a\_place\_for\_a\_bb(bbList, x_{env}, y_{env}, w_{env}, h_{env}, w, h)$
if $w > w_{env}$ *or* $h > h_{env}$ **then**
  return -1,-1,-1,-1
else
  if *bbList* `is empty` **then**
    return $x_{env}, y_{env}, w_{env}, h_{env}$
  else
    $bb \leftarrow head(bbList)$
    $bbList \leftarrow bbList - bb$
    if $bb \cap env$ `is empty` **then**
      return $find\_a\_place\_for\_a\_bb(bbList, x_{env}, y_{env}, w_{env}, h_{env}, w, h)$
    else
      $r_1, r_2, r_3, r_4 \leftarrow get4Rect(bb, x_{env}, y_{env}, w_{env}, h_{env})$
      $i \leftarrow 0$
      while $find\_a\_place\_for\_a\_bb(bbList, x_{r_i}, y_{r_i}, w_{r_i}, h_{r_i}, w, h) == -1, -1, -1, -1$ *and* $i < 4$ **do**
        $i \leftarrow i + 1$
      end while
    end if
  end if
end if

Since the sampling algorithm is defined, let us move to the training of a CNN-based classifier.

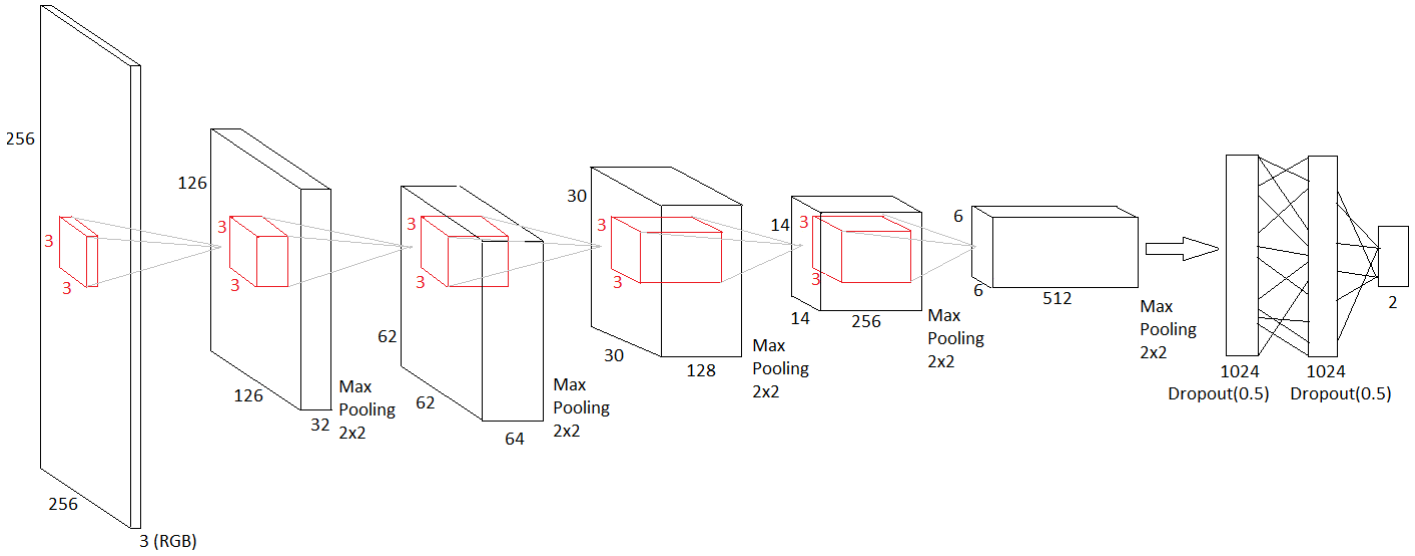

FIGURE 3.16: Architecture of the choosen network
The first five layers are convolutional layers. The last two layers are fully connected.

3.3 Training a CNN-based classifier

3.3.1 Model architecture

General architecture The input presented to the model is a RGB pixel values tensor of the colored database images 256×256. Then, the chosen model is strongly inspired by the "AlexNet" in [14]. The net is indeed constituted by five convolutional layers followed by three fully connected layers (see figure 3.16).

In the first convolutional layer, there are 32 kernels. This number is then doubled in each layer, this means 64 filters in the second layer, 128 in the third, 256 in the fourth and finally 512 in the last layer. Each filter has the same size 3×3 (below 2×2 it is too small and above 7×7 too big to capture all the features).

After each convolutional layer, spatial max pooling of size 2×2 is applied and ReLU is chosen as activation function.

After each fully connected layer which contains 1024 units, half of the units are "dropped out" [30].

There are two final output units since it is a two classification problem. The value of these units can be merged in one vector which can be seen as a probability distribution vector: The component values are real between 0 and 1 and the sum equals to 1. The highest component value gives the corresponding computed class.

The network with approximatively 6.3 millions of units has to learn the set of weights wich minimizes the cost function.

Cost function The cost function the network minimizes during back-propagation is the cross-entropy loss function well-adapted for multiclass classification problems, so in particular binary ones like ours. In the

following part, K is denoted as the number of classes (so $K = 2$ here), N the number of watched instances, p_n the true label vector of size K where every component is null but the one corresponding to the good class, and $\hat{p_n}$ the distribution probabilities vector also of size K actually computed by the network (values between 0 and 1 with a sum equal to 1).

As a reminder, the mean squared difference error also exists and is with the notation above:
$E = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} (p_{n_k} - \hat{p_{n_k}})^2$. But it is not any more the most used error function now.

The cross-entropy loss can be computed as following for the case of the binary classification:
$E = \frac{-1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} (p_{n_k} \, . \, log \, \hat{p_{n_k}} + (1 - p_{n_k}) \, . \, log \, (1 - \hat{p_{n_k}}))$.

$\hat{p_n}$ is computed with the *softmax* function σ which can be seen as a generalization of the logistic function where the returned real value is in the range of $[0, 1]$ (which explains why the softmax function is also called the *exponential normalization*) and the sum is equal to 1. That is how softmax application on the output units enables us to interpret this vector as a probability distribution. The components of this vector are computing as following: $\hat{p_{n_i}} = \sigma(o_i) = \frac{e^{o_i}}{\sum_{k=1}^{K} e^{o_i}}$ where o is the output units vector of size K given by the model with real component values.

3.3.2 Parameters of training

Batch size A *batch* determines the number of frames your network observes before computing the error. What is a good size for a batch? How to fill it ? If the batch is too small, the crops in it won't be representative of the whole dataset. If it is too big, the average on the batch will be too approximative. First we choose 32 but then with 16, because the learning quality stays quite good while the computation time is half-reduced (cf. 3.3.4).

Epoch size An *epoch* is formed from batches. Theoretically, it represents the number of necessary batches to cross all the images of the training set once. In our case, for a given size of batch, the number of positive annotations divided by the batch size should be the epoch size. But in our case, it is adapted : training continues while the validation curve is improving.

Weights initialization The net weights are initialized randomly.

Learning rate The learning rate has been set to 0.03 with a weight decay of value 10^{-4}.

Error computation The network is trained or the error minimization is computed with Stochastic Gradient Descent (SGD) and Nesterov momentum (= with a fixed momentum constant of 0.9) which reduces the risk of getting stuck in a local minimum.

See Appendix B for further explanations.

3.3.3 Pre-processing and data augmentation

3.3.3.1 Data pre-processing

During training, all example images are mean reduced variance normalized to decrease contrast variations, that means to minimize the effect of different lighting conditions, as advised in [3–5]. The same is performed during testing.

3.3.3.2 Data augmentation

To prevent overfitting, first, we try to merge as many large datasets as we can find to add some diversity in the data (those described in section 3.1.2 . Secondly we operate some data augmentation. For instance, like in [3, 14], to dispose of more labeled data in our training set, we add the left-right reflections of each crops (also called mirror operation). See section 3.4.2.2 for others detailed made choices on the matter.

3.3.3.3 Boostrapping

Boostrapping consists in searching exhaustively negative training frames for false positive and feeding the network an other time with those "hard examples". This method is widely applied [3, 5, 8, 17, 22, 40] and so it has been done in this work.

3.3.4 Work on computation time optimization

Training a deep network can be very long. So a particular work has been made to optimize the network training computation time by parallelization of code. First two threads have been used to accelerate the waiting time between two batches. Indeed, the batch data loading is quite long. So a slave thread has been set to fill an other batch while the master thread computes the network weights update for the current batch.

Then, since it was not enough, GPU computation has been involved for the network weights forward and backward primitives processes (the model was only computed on the CPU before). A Graphics Processing Unit (GPU) is an electronic circuit for accelerating the display of images but modern GPUs are although very used in image processing when algorithms are designed for processing large blocks of data in parallel.

Thereby the computation time for loading a batch has been reduced. As three GPUs were available for this project, three model trainings could be started at the same time without increasing the time computation for each one.

The table 3.4 reveals the progress made by parallelization.

	Action	Weights update loc.	Batch size	Comp. time
Sequential	Initial state	CPU	32	2s 30 ms
Parallel	2 threads	CPU	32	1s 40 ms
	2 threads + GPU	GPU	32	1s 20 ms
		GPU	16	500 ms
		GPU	8	300 ms

TABLE 3.4: Conputation time optimization for a network with 3 convolutional layers

3.3.5 Validation and learning rate policy application

An equal learning rate for all layers has been used and adjusted manually through training. The followed heuristic is to divide the learning rate by 10 when the validation error stopped improving with the current learning rate (plateaued), like in [14].

Our model and propositions have been presented. In the next section, the results will be shown and evaluated.

3.4 Results and evaluation

3.4.1 Sampling algorithm

The sampling algorithm plays a key role for the quality of the obtained classification accuracy. Thus two methods for measuring the randomness of the drawn crops have been established:

- First, *histograms* to show that each positive annotation and each frame for negative crops are correctly uniformely drawn. This experiment enabled to see that at the begining too much constraints were put on the positive wanted annotations (size, distance from the bounds...). So too few positive annotations were available for drawing which leads to a very soon overfitting. Consequently drawing parameters have been righlty adjusted to have naturally more different drawn samples from the dataset.
- The weakness of the histograms was that the uniformity of negative crops drawing from all frames could not be judged. So this fact has been checked with *heat maps*. For each frame, (1) a *heat map* is created of the frame same size and all pixels are set with value zero; (2) 1000 random crops are drawn from this frame. Each time a pixel of the frame is contained in a drawn crop, the corresponding pixel value in the heat map is incremented. The higher its value, the whiter the pixel.

Histograms The following histograms have been obtained after the classifier learned from arbitrary 10 epochs of 50 16-sized batches. The training set is exclusively for this experiment the USA training set which contains as a reminder, 4250 frames (2000 approximatively of them are negative) and 5080 person class annotations. So 8000 crops have been submitted to the model. Approximatively half of them are positive crops. Two distributions depending on the drawing parameters are shown with figures 3.17 and 3.18 (cf.

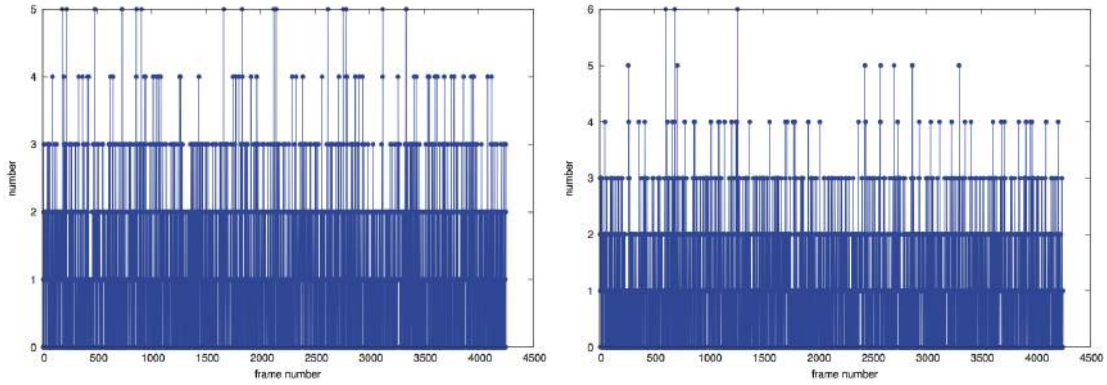

FIGURE 3.17: Distribution of selected frames for negative crops, config. 1 & 2
LEFT : $w\_bb\_min = 5$, $h\_bb\_min = 20$, $method\_rand\_hw = 1$, $method\_rand\_bb = 2$, $jaccard\_index = 0.1$, $ratio\_w\_bb\_max = 1$, $ratio\_h\_bb\_max = 1$. RIGHT : $w\_bb\_min = 10$, $h\_bb\_min = 30$, $method\_rand\_hw = 1$, $method\_rand\_bb = 2$, $jaccard\_index = 0$, $ratio\_w\_bb\_max = 0.5$, $ratio\_h\_bb\_max = 0.5$.

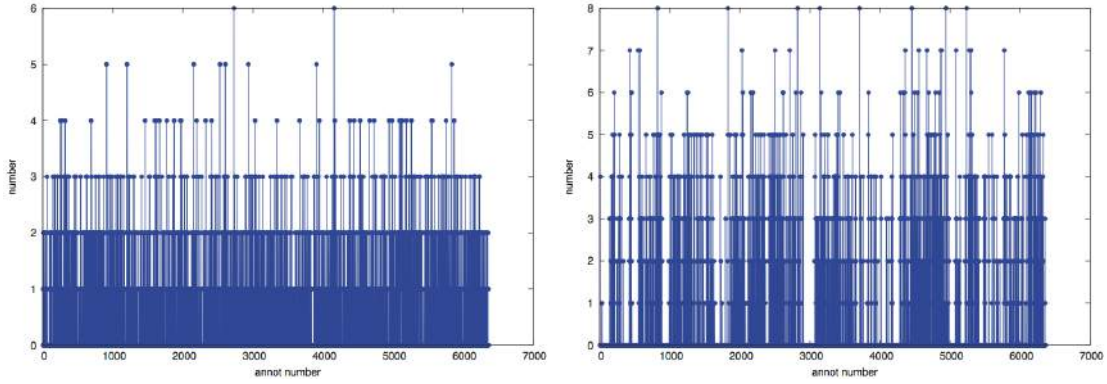

FIGURE 3.18: Distribution of positive annotations, config. 1 & 2
LEFT : $w\_bb\_min = 5$, $h\_bb\_min = 20$, $allowed\_dist\_from\_bounds = -1$. RIGHT : $w\_bb\_min = 10$, $h\_bb\_min = 30$, $allowed\_dist\_from\_bounds = 0.05$.

section 3.2.1 for the parameters meaning). Figure 3.17 shows how many times a frame is selected to find a negative crop. Theoretically, this should be a uniform distribution. Some frames are less selected than others since they are already full of annotations and it could be difficult to find a crop respecting the constraints on overlap. For respectively configuration 1 and 2, 2637 and 2636 different frames for drawing a negative crop have been selected. Figure 3.18 shows how many times a positive annotation is drawn to be shown to the model. The distribution is uniform too. For respectively configuration 1 and 2, 2687 and 1485 different positive annotations have been drawn.

Heat maps Here are the results of negative crops drawn from negative and positive frames from the USA training set [35]. Figure 3.19 shows the results for a positive frame (so which contains positive annotation, here only the person class is concerned) with no overlap between negative and positive crops controlled by the method 2 (the four rectangles method of algorithm 3) and a jaccard index that could not be higher than 0. Figure 3.20 presents the easy case where the frame is negative. Finally, figure 3.21 displays the results for the case of a positive frame with overlapping allowed (jaccard index < 0.1).

For figure 3.19, dark spaces can be seen created by the distinct four rectangles around the positive annotations. One of the four rectangles is particularly white (bottom, right). This does not question the randomness quality

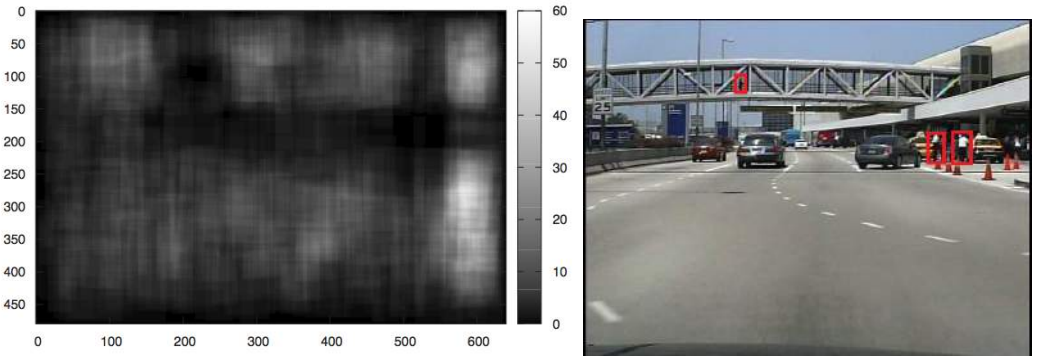

FIGURE 3.19: Negative crops drawing from a positive frame with no overlap allowed
Left the heat map, right the original frame.

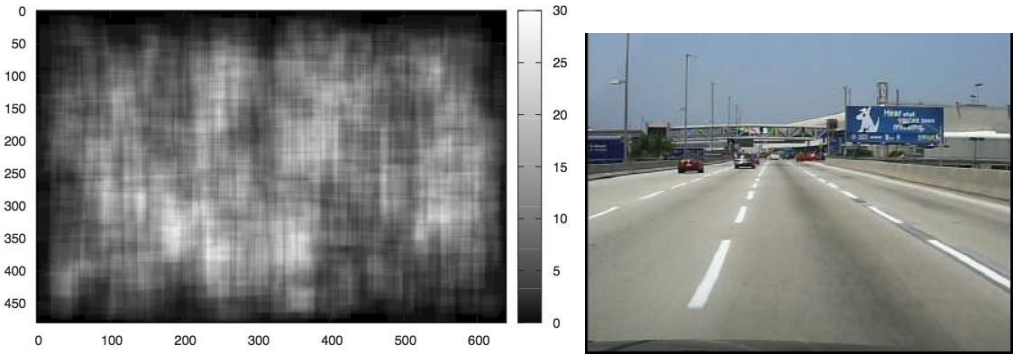

FIGURE 3.20: Heat map : Negative crops drawing from a negative frame.
Left the heat map, right the original frame. The crops are ramdomly drawn from all the frame.

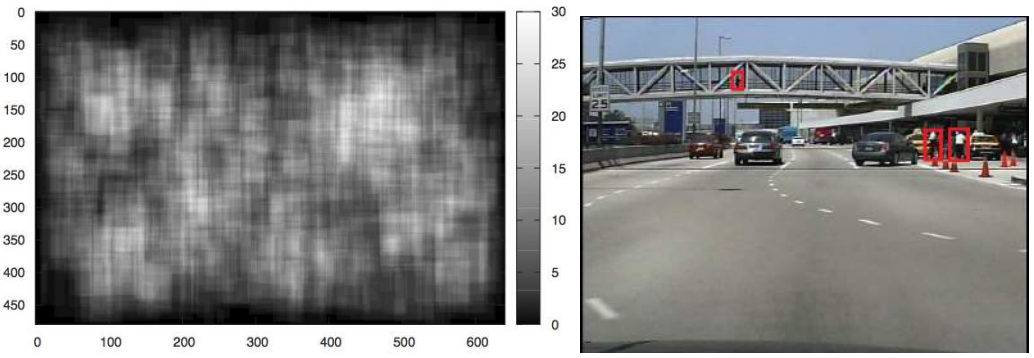

FIGURE 3.21: Heat map : Negative crops drawing from a positive frame with jaccard index < 0.1
Left the heat map, right the original frame. When overlap is allowed in the positive frame the results are hopefully similar to those for a negative frame.

of the drawing. Indeed, each of the four rectangle have the same probability to be selected for the drawing but in a smaller space, the probability is higher to have negative crops overlapping with other negative ones. The drawback of avoiding overlapping with positive annotations though is to block four directions around the annotation. So in practice, overlap is allowed.

3.4.2 Results on classifier

Some experiments have been led to determine the right parameters for the learning phase by studying:

- influence of the learning rate value

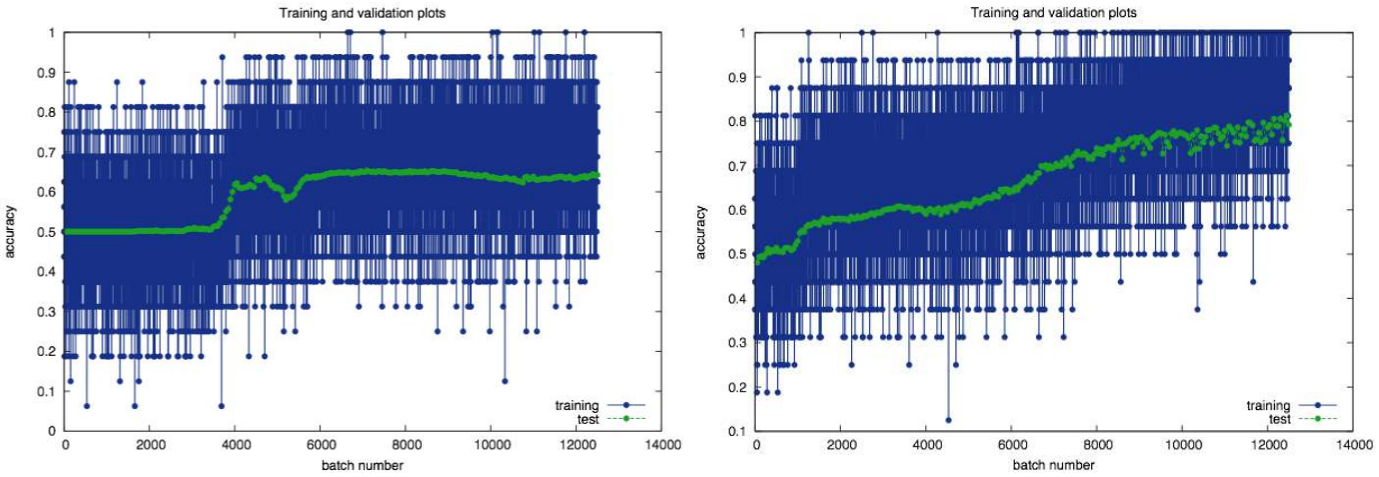

FIGURE 3.22: Learning rate : Training and validation plots for $lr = 0.01$ (left) and $lr = 0.02$ (right)

- role of data augmentation methods on overfitting
- impact of the network depth
- influence of the pre-training on accuracy

3.4.2.1 Influence of the learning rate

The training (blue) and validation (green) plots are shown for four values of learning rate (lr) 0.01, 0.02, 0.03 and 0.05 in figures 3.22 and 3.23. The network is the one described above in section 3.16. The learning is made on 250 epochs of 50 batches with size 16 exclusively on the USA training set. The well-classified image rate is plotted against the current batch number. A validation is made every epoch on the official USA test set.

Interpretation With a learning rate below 0.03, the validation plot has difficulties to take off in comparison with the training curve. The rule is to take the smallest learning rate enabling learning because the highest the learning rate, the less accurate the training and the sooner overfitting occurs. Indeed, for respectively $lr = 0.02$, $lr = 0.03$, $lr = 0.05$, overfitting approximatively occurs at the 8000-th, 3850-th batch, 3500-th batch. The learning rate is fixed with $lr = 0.03$ for this network configuration since among the three values of learning rate $lr = 0.02$, $lr = 0.03$, $lr = 0.05$, the validation curve for $lr = 0.03$ is the one following the best the training curve evolution.

3.4.2.2 Role of data augmentation methods

Still with $lr = 0.03$, overfitting is reached around the 3850-th batch (but later than for $lr = 0.05$ around the 3500-th batch) while the accuracy in validation is below 0.9. So different data augmentation methods have been tested to delay overfitting:

- mirror reflections or horizontal flipping which actually doubles the number of available data.

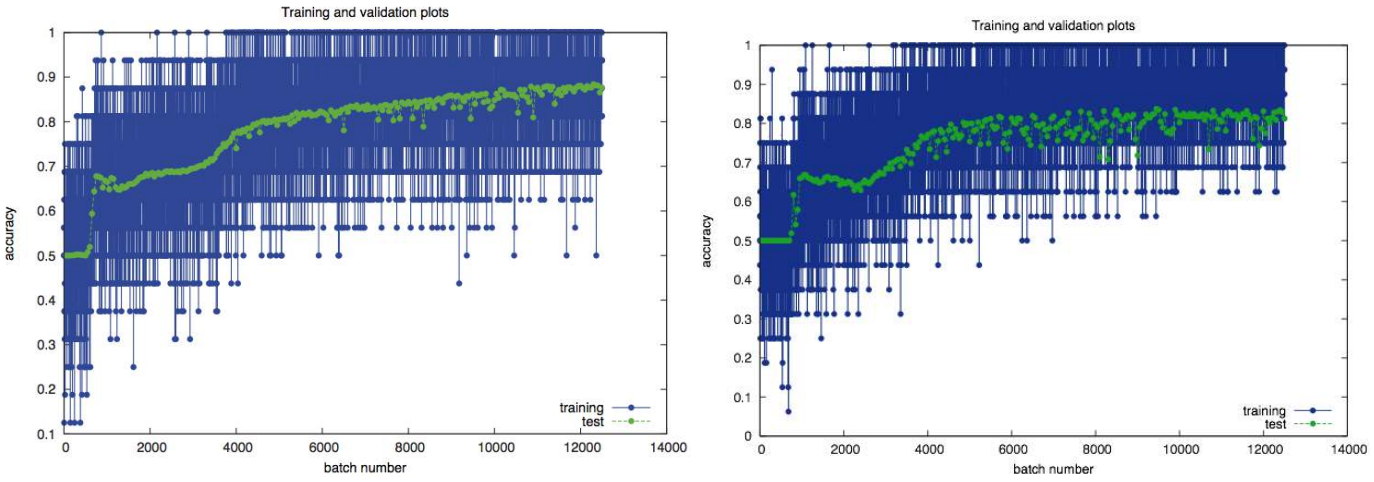

FIGURE 3.23: Learning rate : Training and validation plots for $lr = 0.03$ (left) and $lr = 0.05$ (right)

- translation: from the original crop has been made a new one with little horizontal and/or vertical shifts which represent maximally 10% of the width and height of the original crop.
- aspect ratio modifications: it is an expansion (ratio above 1) or a compression (ratio below 1) of the original crop. A ratio for the width and the height have to be fixed. In our case: $(1, 1)$ (no change), $(2, 2)$, $(1, 2)$, $(2, 1)$ are available. $(2, 1)$ means that from the original crop, the width is streched until it reaches twice the original width, while the height remains the same.
- little rotations : the original crop is rotated of $\pm$ 0, 0.03, 0.07, 0.13, or 0.26 *rad* with respective probabilities 0.3, 0.3, 0.2, 0.1, 0.1.
- boostrapping with hard negatives examples : a set of 2001 hard negatives has been made after training during 4000 batches of size 16

As those computations are not very costly, they are not stored in memory. Other tentatives have been made for data augmentation like (1) colorimetry changes or (2) different croppings around the positive annotations but the results were irrelevant. Finally, (3) some warping which consists in associating one of the eight neighbors to a point in the image leads to a kind of blurring and gives results as good as those from the previous cited data augmentation methods but not better (further experiments are currently performed). See figures 3.24, 3.25, 3.26 for results of the progressive addition of new data augmentation methods and figures 3.27 and 3.28 for the own effect of each method.

Interpretation of figures 3.24, 3.25, 3.26: The table 3.5 shows from which batch the network begins to overfit on data according to the used data augmentation methods (1 stands for a use of the method, 0 otherwise). First, all those methods enable to delay overfitting even if it is less obvious for the aspect ratio transformations. Nevertheless, bootstrapping which submits to the network hard negative samples with probability 0.5 when a negative crop is expected, is incredibly more efficient than the others by multiplying by 2.6 the delay time before overfitting in comparison with the results without data augmentation methods.

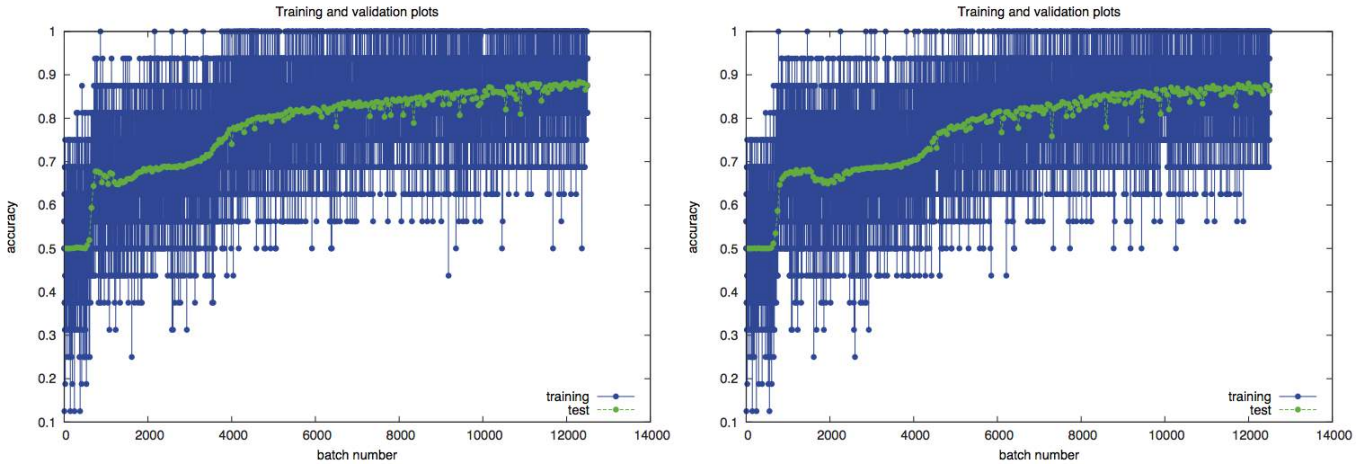

FIGURE 3.24: Data augmentation : Training and validation plots for $lr = 0.03$, 5 convolutional layers, no data augmentation (left) and mirror reflections (right)

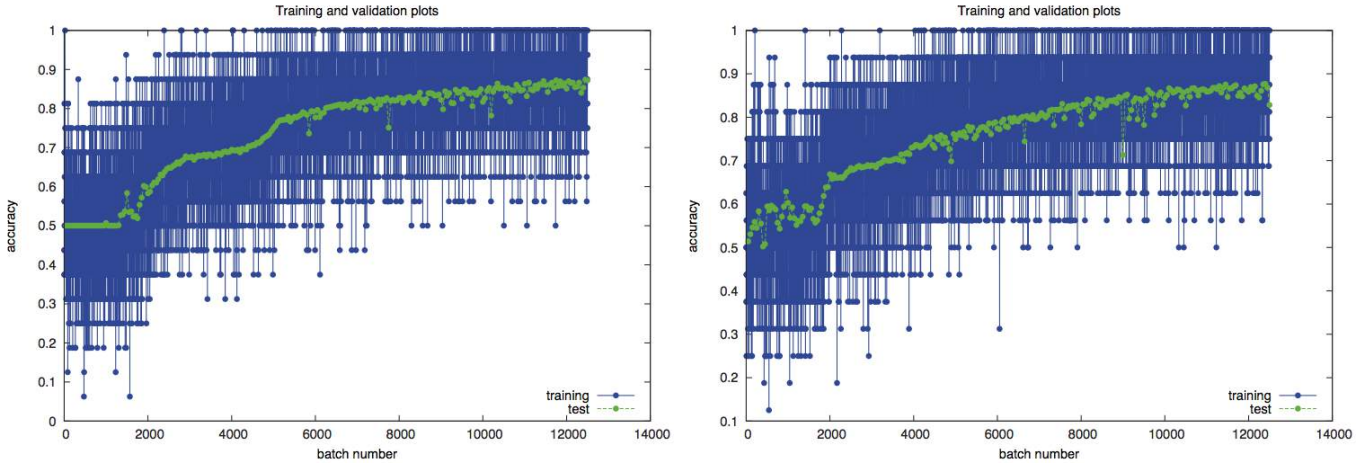

FIGURE 3.25: Data augmentation : Training and validation plots for $lr = 0.03$, 5 convolutional layers, mirror + shift (left) and mirror + shift + aspect ratio transformations (right)

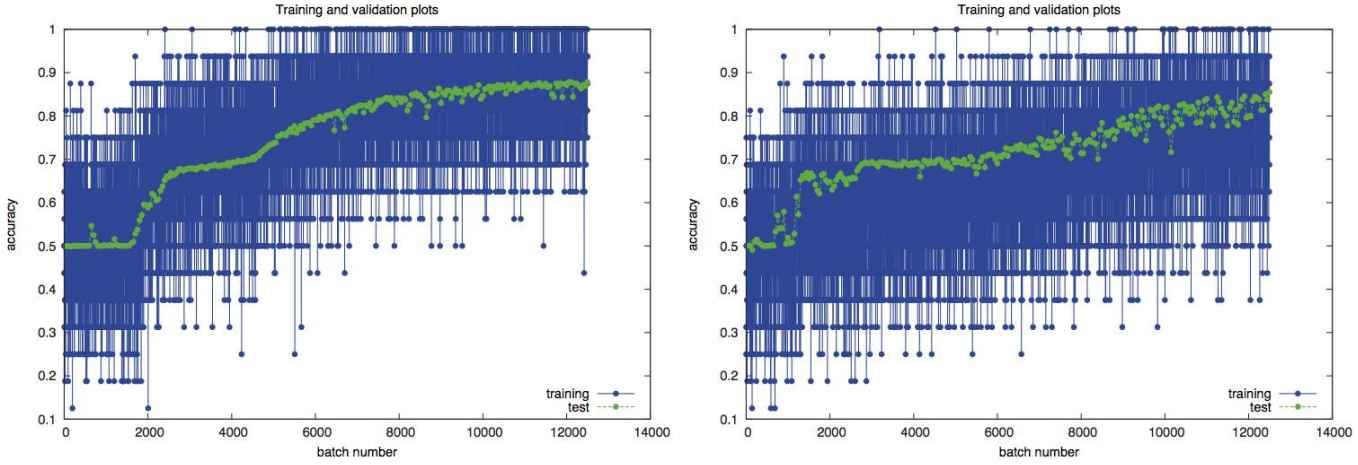

FIGURE 3.26: Data augmentation : Training and validation plots for $lr = 0.03$, 5 convolutional layers, mirror + shift + aspect ratio + little rotations transformations (left) and mirror + shift + aspect ratio + little rotations transformations + boostrapping (right)

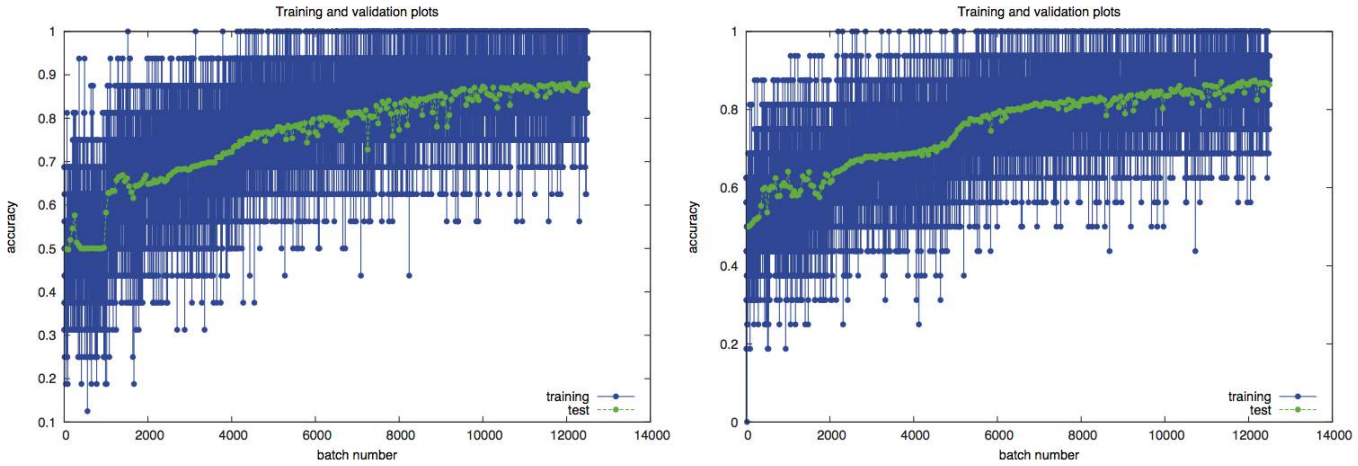

FIGURE 3.27: Data augmentation : Training and validation plots for $lr = 0.03$, 5 convolutional layers, only aspect ratio transformations (left) and shifting (right)

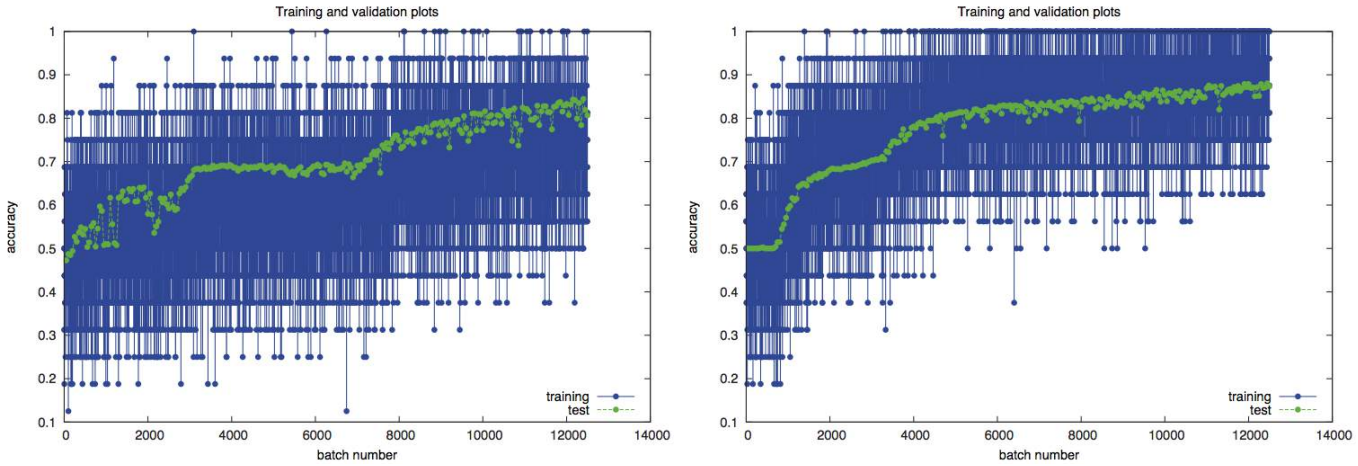

FIGURE 3.28: Data augmentation : Training and validation plots for $lr = 0.03$, 5 convolutional layers, only hard negatives (left) and rotations (right)

N	Mirror	Shift	Aspect ratio	Rotations	Bootstrapping	Overfitting from batch
0	0	0	0	0	0	≈ 3850
1	1	0	0	0	0	4000
2	1	1	0	0	0	4500
3	1	1	1	0	0	4000
4	1	1	1	1	0	5000
5	1	1	1	1	1	10000
6	0	0	1	0	0	≈ 4100
7	0	1	0	0	0	5000
8	0	0	0	0	1	8000
9	0	0	0	1	0	4000

TABLE 3.5: Impact of data augmentation methods on overfitting

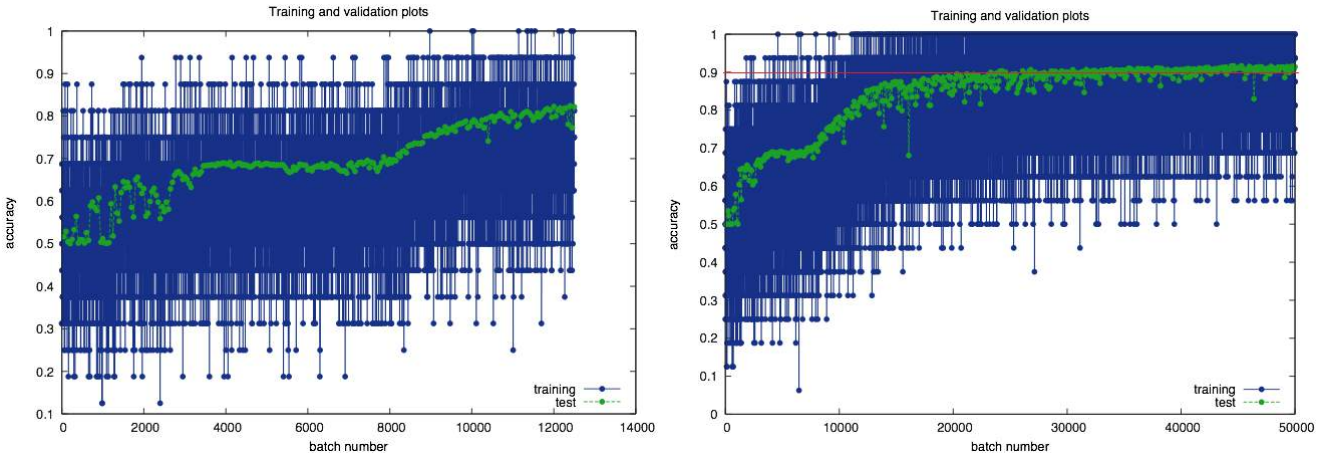

FIGURE 3.29: Learning rate policy : Training and validation plots for $lr = 0.03$, 5 convolutional layers with mirror + shift + aspect ratio + rotations transformations + hard negatives + learning rate policy at 2500-th batch on totally 12500 batches (left) and 50000 batches (right)

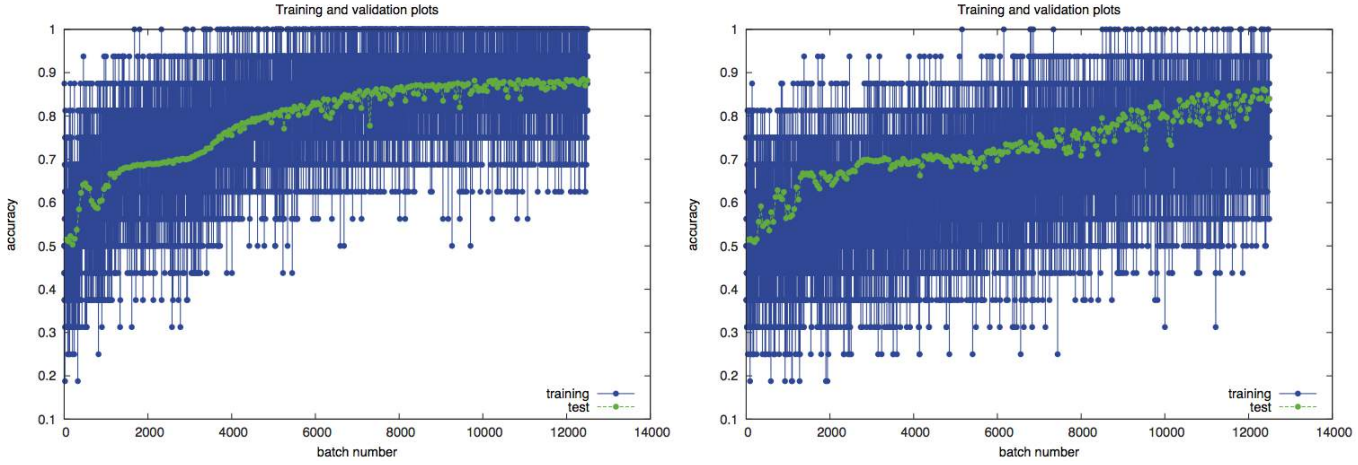

FIGURE 3.30: Network depth : Training and validation plots for $lr = 0.03$, 4 convolutional layers with no data augmentation (left) and mirror + shift + aspec ratio + rotations transformations + hard negatives (right)

Further improvements with learning rate policy Following the preceding study on the learning rate influence, a learning rate policy is further applied to prevent overfitting which consists to divide the learning rate by 10 approximatively before a plateau appears on the validation plot. Around the 2500-th batch for the right plot with all data augmentation methods in figure 3.26 a kind of plateau occurs. In the resulting figure 3.29, the overfitting is obviously delayed compared to the same configuration without learning rate policy (from the 10000-th batch to approximatively the 11000-th batch). Moreover, with enough time, the network is enabled to classify the images with more than 0.9 of accuracy. It was tried also after the 3000-batch to divide by 10 one again the learning rate but it did not give improvements.

3.4.2.3 Impact of the network depth

The results on impact of the network depth have been obtained with a network architecture of 5 convolutional layers after increasing progressively the layers number. See figures 3.30 and 3.31 for the previous results obtained with 4 and 3 convolutional layers.

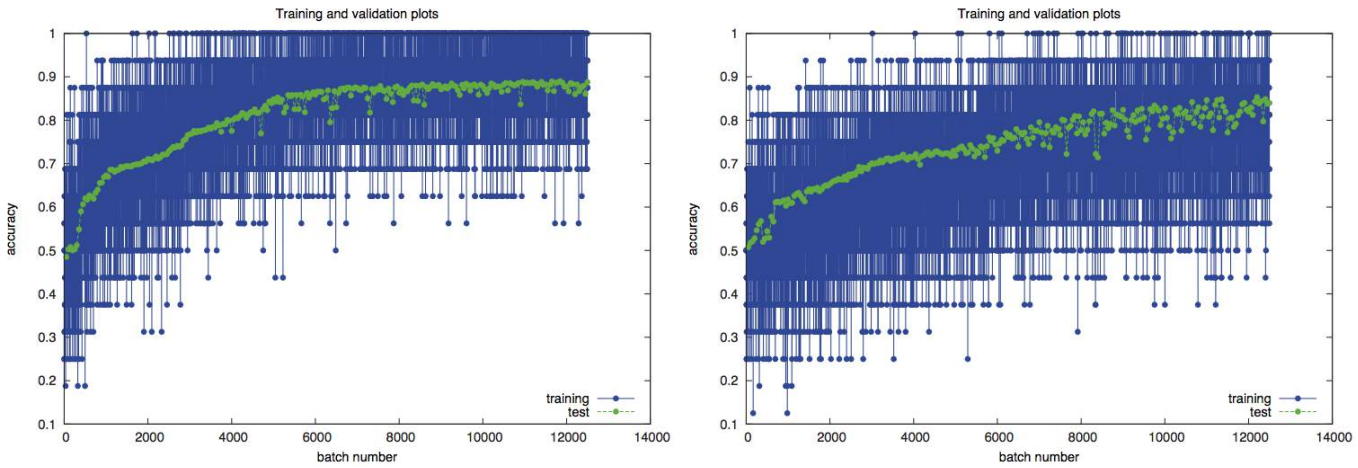

FIGURE 3.31: Network depth : Training and validation plots for $lr = 0.03$, 3 convolutional layers with no data augmentation (left) and mirror + shift + aspect ratio + rotations transformations + hard negatives (right)

Interpretation With no data augmentation, there is no meaningful difference between 5, 4, or 3 convolutional layers. With all the data augmentation methods, the validation curve slope of the network with 3 layers is lower than for the ones with 4 or 5 layers. Then the difference is not significant between a network with 4 layers or 5.

3.4.2.4 Pre-training influence

The following experiment has been led:

- First a hard negative sample of size 7282 has been made with every data but the ones from the USA dataset.

- Second, a first model has been trained on every data but the ones from the USA dataset with $lr = 0.3$ and all data augmentation methods descrived previously.

- Finally, an other model is trained on USA data training set only with $lr = 0.001$ and all data augmentation methods descrived previously on 250 epochs of 50 batches of size 16.

See the results in figure 3.32. This enables to reach 0.90% of accuracy but not above, sadly.

3.5 Discussion

These results have to be compared with current state of the art accuracy. The pedestrian classification performance is actually measured with the ROC curve specially designed for illustrating the performance of a binary classifier. The curve is created by plotting the true positive rate against the false positive rate. At each new sample from the test set, the point (number of negative crops classified as positive on the number of negative crops, number of positive crops well-classified on the number of positive crops) is plotted. As a reminder, the diagonal which represents a random classification divides the ROC space this way : Points

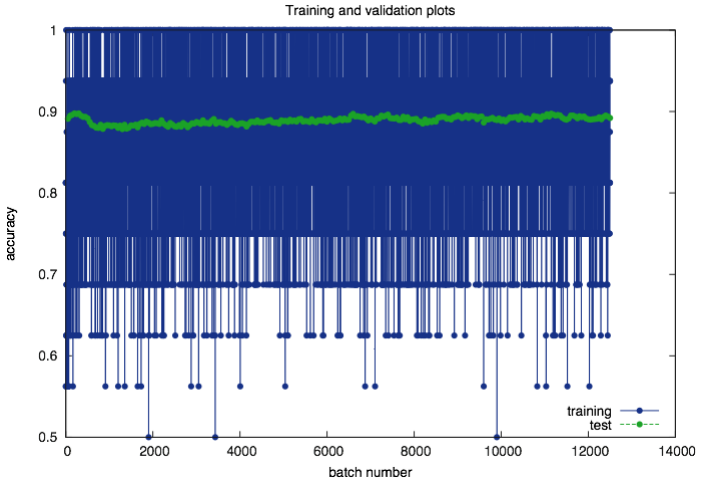

FIGURE 3.32: Pre-training : Training and validation plots for $lr = 0.05$, 5 convolutional layers with mirror + shift + aspect ratio + rotations transformations + hard negatives after pre-training

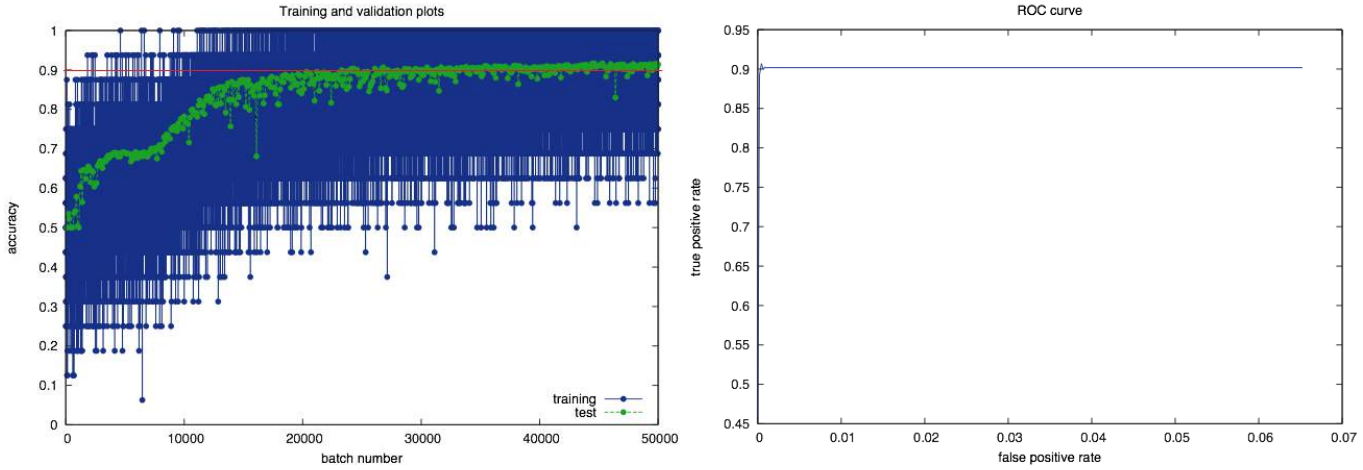

FIGURE 3.33: ROC curve for $lr = 0.03$ with the corresponding training and validation plots, 5 convolutional layers with mirror + shift + aspect ratio + rotations transformations + hard negatives + learning rate policy on the 2500-th batch

above the diagonal are for good classification results, otherwise for poor results. See 3.33 for the best obtained results.

State of the art accuracy reaches almost 1, see figure 3.34.

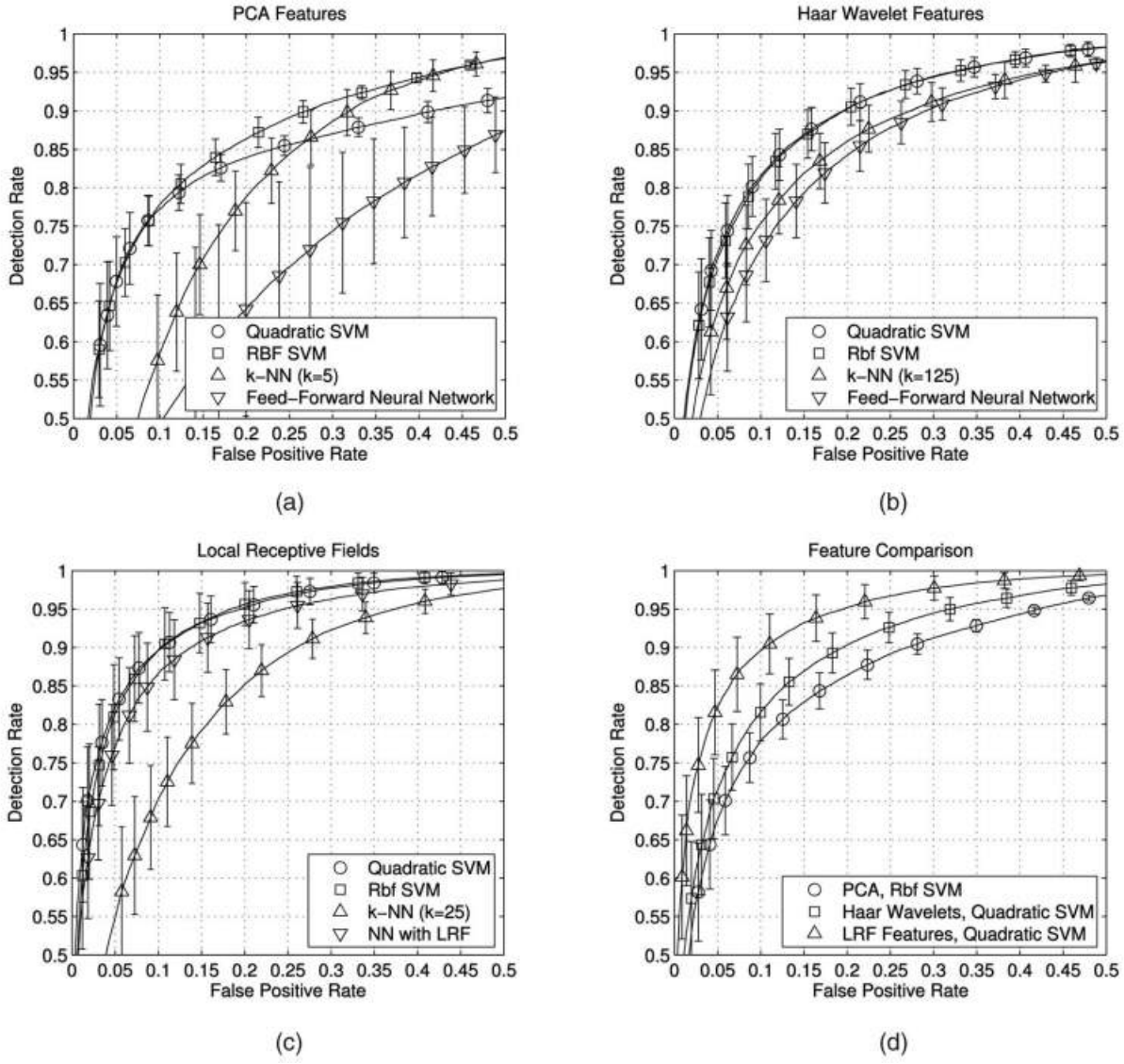

FIGURE 3.34: ROC curves examples from [1]: "A comparison of different feature extraction and classification methods. Performance of different classifiers on (a) PCA coefficients, (b) Haar wavelets, and (c) Local Receptive Field (LRF) features. (d) A performance comparison of the best classifiers for each feature type."

Chapter 4

Conclusion and future directions

Here is the end of this Master thesis internship report. As this work falls within an initial more ambitious project, all the steps were not expected to be done and to obtain the best possible results for each steps was a necessary goal. That is why eventually the final detection step was not implemented in this internship but great care has been given to the sampling algorithm instead, since the quality of the classification and detection depend on the dataset one. This was enabled among others by the tests for checking the crops' randomness and allocation in all images. To beat the state of the art for pedestrian detection, we set for the classification task a threshold to exceed before to move to the detection: We wanted to obtain 98% of well-classified images rate. We are currently around 93% for the validation by training with the USA training set and validating on the USA test set, which still can be improved!

To conlude this report, during this internship, some datasets have been merged to build a consistant positive annotations database for pedestrian detection in images. Then a sampling algorithm has been implemented which carefully draw positive and negative crops from images respecting constraints in parameters. Finally, CNN-based classifiers have been trained for the pedestrian classification task whose results have been improved by data augmentation methods (up to 93% of well-classified images).

Different perspectives could now further improve this result: (1) merge more and more datasets, (2) improve the quality of the datasets (wrong or missed annotations have been noticed), (3) perform other data augmentation methods from the elastic transformations family like perspective distorsion transformations, (4) why not trying to use synthetic positive and negative samples from virtual images which are so much easier to obtain, like in [41] ? An other relevant point is to use all the information given by video sequences, this means the temporal structure, the (5) motion information which can help to identify occluded pedestrian. Finally, (6) part-based models showed great performance [22]. With one or some of these tricks, the expected challenging well-classified images rate of 98% could be hopefully reached.

Appendix A

How to apply convolution to images?

Algorithm 4 Convolution algorithm

```
1: for each image row in output image do
2:   for each pixel in image row do
3:     accumulator ← 0
4:     for each kernel row in kernel do
5:       for each element in kernel row do
6:         if element position corresponding to pixel position then
7:           multiply element value corresponding to pixel value
8:           add result to accumulator
9:         end if
10:      end for
11:    end for
12:    set accumulator to output image pixel
13:  end for
14: end for
```

Appendix B

Back-propagation

This appendix is about two reminders concerning neural network back-propagation that aims to find the set of weights which minimizes the error between the computed output and the real target (see section 3.3.1 for further information about the error function) : (1) propagation or network output computation and (2) network weights update (*back*-propagation).

B.1 Propagation

The *forward* propagation is complete when every activation function outputs have been computed. Let be x_i^0 an input unit of the input layer 0, y_i^l a final network output unit of layer l and φ the last layer activation function:

$$y_i^l = f(x_i^0) = \varphi\left(\sum_{j\in A_j} w_{ij}^l\, y_j^{l-1}\right) = \varphi\left(\sum_{j\in A_j} w_{ij}^l\, g_i^{l-1}(x_i^0)\right)$$

where A_j is the set of the anterior nodes of y_i^l, w_{ij}^l are the weight between units i and j at the layer l and g_i^{l-1} hides a collection of functions corresponding to the computed output of the previous layers.

B.2 Weights update

B.2.1 Stochastic Gradient Descent (SGD) with learning rate and momemtum

New values for the weights are determined so that the error function is minimized. This optimization is led by the Stochastic Gradient Descent (SGD). Here is a reminder of the SGD method principle: if x_T is a local minimum of a multivariate F defined and differentiable in the neighborhood of the point $x(t)$ with $x(t+1) = x(t) - \gamma(t)\ \nabla F(x(t))$, $\gamma(t)$ very small, $t \geq 0$, so $F(x(t)) \geq F(x(t+1)) \geq ... \geq F(x_T)$ and the sequence $(x(t))$ converges to the local minimum.

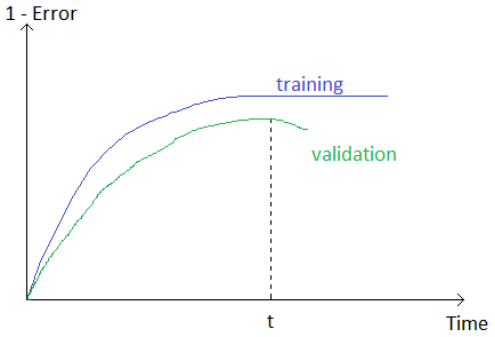

FIGURE B.1: Training and validation plots to illustrate overfitting

In the same way, for a particular weight w_{ij} at a certain layer (weight for the i-input to the j-th unit) at the time t, $w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t+1)$ with $\Delta w_{ij}(t+1) = -\mu \frac{\partial E_i}{\partial w} + m \Delta w_{ij}(t)$ where μ is the learning rate, E the error to minimize and m the momemtum parameter ($\Delta w(t) = w(t) - w(t-1)$).

Remarks (1) The weight update is proportional to the negative gradient error plus the previous weight change because the sign of the error gradient indicates where the error is increasing, so the weights have to be updated in the opposite direction.

(2) The higher the learning rate is, the faster the network trains but the less accurate the training is.

(3) The momentum parameter must satisfy $0 \leq m < 1$. The value is often set to 0.9 (the Nesterov momentum). The momentum enables to reduce the oscillations during the stochastic gradient descent.

B.2.2 Weight decay

According to figure B.1, overfitting occurs when the validation performance decreases (here from time t) and shows a high degree of curvature which translates a too huge number of parameters in the network. A way to make this function smoother is to add a penalty to the error function proportional to the square sum of all weights: $\tilde{E} = E + \frac{1}{2} \lambda \sum_k w_k^2$ and the weight update becomes: $w_{ij}(t+1) = w_{ij}(t) - \mu \frac{\partial E}{\partial w} + m \Delta w_{ij}(t) - \mu \lambda w_{ij}(t)$ where λ is the regularization parameter which causes the weight to decay in proportion to its size and so it is called *weight decay*. Consequently, parameters with high value tend to approach zero and that effectively reduces the number of (active) parameters and overfitting.

Bibliography

[1] S. Munder and D. M. Gavrila. An experimental study on pedestrian classification. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 28(11):1863–1868, 2006.

[2] R. Benenson, M. Omran, J. Hosang, , and B. Schiele. Ten years of pedestrian detection, what have we learned? 2014.

[3] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. pages 886–893, 2005. doi: 10.1109/CVPR.2005.177. URL http://dx.doi.org/10.1109/CVPR.2005.177.

[4] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. pages 511–518, 2001.

[5] Paul Viola, Michael J. Jones, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. *Int. J. Comput. Vision*, 63(2):153–161, July 2005. ISSN 0920-5691. doi: 10.1007/s11263-005-6644-8. URL http://dx.doi.org/10.1007/s11263-005-6644-8.

[6] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60 (2):91–110, November 2004. ISSN 0920-5691. doi: 10.1023/B:VISI.0000029664.99615.94. URL http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94.

[7] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. pages 23–37, 1995. URL http://dl.acm.org/citation.cfm?id=646943.712093.

[8] Piotr Dollar, Zhuowen Tu, Pietro Perona, and Serge Belongie. Integral channel features. pages 91.1–91.11, 2009. doi:10.5244/C.23.91.

[9] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. pages 318–362, 1986. URL http://dl.acm.org/citation.cfm?id=104279.104293.

[11] Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. pages 396–404, 1990.

[12] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. pages 2278–2324, 1998.

[13] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006. ISSN 0899-7667. doi: 10.1162/neco.2006.18.7.1527. URL http://dx.doi.org/10.1162/neco.2006.18.7.1527.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. pages 1097–1105, 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[15] J. Deng qnd A. Berg, A. Khosla S. Satheesh, H. Su, and L. Fei-Fei. Imagenet large scale visual recognition competition 2012 (ilsvrc2012). URL http://www.image-net.org/challenges/LSVRC/2012/.

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. 2009.

[17] Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala, and Yann LeCun. Pedestrian detection with unsupervised multi-stage feature learning. *CoRR*, abs/1212.0142, 2012. URL http://arxiv.org/abs/1212.0142.

[18] Wanli Ouyang and Xiaogang Wang. Joint deep learning for pedestrian detection. pages 2056–2063, 2013. doi: 10.1109/ICCV.2013.257. URL http://dx.doi.org/10.1109/ICCV.2013.257.

[19] Lei Wang and Baochang Zhang. Boosting-like deep learning for pedestrian detection. *CoRR*, abs/1505.06800, 2015. URL http://arxiv.org/abs/1505.06800.

[20] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013. URL http://arxiv.org/abs/1311.2524.

[21] Koen E. A. van de Sande, Jasper R. R. Uijlings, Theo Gevers, and Arnold W. M. Smeulders. Segmentation as selective search for object recognition. pages 1879–1886, 2011. doi: 10.1109/ICCV.2011.6126456. URL http://dx.doi.org/10.1109/ICCV.2011.6126456.

[22] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(9):1627–1645, September 2010. ISSN 0162-8828. doi: 10.1109/TPAMI.2009.167. URL http://dx.doi.org/10.1109/TPAMI.2009.167.

[23] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. pages 92–101, 2010. URL http://dl.acm.org/citation.cfm?id=1886436.1886447.

[24] Geoffrey E. Hinton and Vinod Nair. Rectified linear units improve restricted boltzmann machines. 2010.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. URL http://arxiv.org/abs/1502.01852.

[26] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout Networks. *ArXiv e-prints*, February 2013.

[27] Patrice Y. Simard, Dave Steinkraus, and John C. Platt. J.c.: Best practices for convolutional neural networks applied to visual document analysis. pages 958–963, 2003.

[28] Jurgen Schmidhuber. Multi-column deep neural networks for image classification. pages 3642–3649, 2012. URL http://dl.acm.org/citation.cfm?id=2354409.2354694.

[29] Dan C. Ciresan, Ueli Meier, Jonathan Masci, Luca M. Gambardella, and Jurgen Schmidhuber. High-performance neural networks for visual object classification. 2011.

[30] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL http://arxiv.org/abs/1207.0580.

[31] M. Enzweiler and D.M. Gavrila. Dense stereo-based roi generation for pedestrian detection. 2009.

[32] A. Ess, B. Leibe, K. Schindler, and L. Van Gool. A mobile vision system for robust multi-person tracking. 2008.

[33] N. Dala and B. Triggs. Histograms of oriented gradients for human detection. 2005.

[34] C. Wojek, S. Walk, and B. Schiele. Multi-cue onboard pedestrian detection. 2009.

[35] P. Dollar, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: a benchmark. 2009.

[36] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. URL http://arxiv.org/abs/1405.0312.

[37] Yubin DENG, Ping Luo, Chen Change Loy, and Xiaoou Tang. Pedestrian attribute recognition at far distance. pages 789–792, 2014. doi: 10.1145/2647868.2654966. URL http://doi.acm.org/10.1145/2647868.2654966.

[38] Stanley Michael Bileschi. Streetscenes: Towards scene understanding in still images. 2006.

[39] Javier Marín, David Vázquez, Antonio M. López, Jaume Amores, and Ludmila I. Kuncheva. Occlusion handling via random subspace classifiers for human detection. *IEEE T. Cybernetics*, 44(3):342–354, 2014. doi: 10.1109/TCYB.2013.2255271. URL http://dx.doi.org/10.1109/TCYB.2013.2255271.

[40] Sakrapee Paisitkriangkrai, Chunhua Shen, and Anton van den Hengel. Strengthening the effectiveness of pedestrian detection with spatially pooled features. pages 546–561, 2014.

[41] Javier Marín, David Vázquez, David Gerónimo, and Antonio M. López. Learning appearance in virtual scenarios for pedestrian detection. pages 137–144, 2010. doi: 10.1109/CVPR.2010.5540218. URL http://dx.doi.org/10.1109/CVPR.2010.5540218.