POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione

Corso di Laurea in Ingegneria Informatica

CONVOLUTIONAL NEURAL NETWORK BASED METHOD FOR PEDESTRIAN DETECTION

Relatore: Prof. Marco TAGLIASACCHI
Correlatore: Ing. Luca BONDI

Tesi di Laurea di:

Denis TOME'

Matr. 817538

Anno Accademico 2014 / 2015

Contents

List of Figures

List of Tables

Abbreviations

CNN	**C**onvolutional **N**eural **N**etwork
RCNN	**R**egions with **C**onvolutional **N**eural **N**etwork
ReLU	**Re**ctified **L**inear **U**nit
DL	**D**eep **L**earning
NN	**N**eural **N**etwork
SVM	**S**upport **V**ector **M**achine
ACF	**A**ggregated **C**hannel **F**eatures
LDCF	**L**ocally **D**ecorrelated **C**hannel **F**eatures
HOG	**H**istogram of **O**riented **G**radients
LAMR	**L**og **A**verage **M**iss **R**ate
IoU	**I**ntersection **o**ver **U**nion
GPU	**G**rahpical **P**rocessing **U**nit
LDA	**L**inear **D**iscriminant **A**nalysis
CTA	**C**ompress **T**hen **A**nalize
ATC	**A**nalyze **T**hen **C**ompress
BB	**B**ounding **B**ox
TP	**T**rue **P**ositives
FP	**F**alse **P**ositives
FN	**F**alse **N**egatives
FPPI	**F**alse **P**ositive **P**er **I**mage
ROC	**R**eceiver **O**perating **C**haracteristic

Sommario

Nell'ambito dell' intelligenza artificiale, uno degli obbiettivi che negli ultimi tempi si è cercato di raggiungere, è la creazione di sistemi migliori degli esseri umani (o comunque comparabili), nei sensi primari come vista e udito.

Lo stato dell'arte nell'ambito del riconoscimento visivo è rappresentato da algoritmi basati su Reti Neurali Convolutive. In particolare, queste hanno dimostrato essere molto più accurate rispetto ad altri metodi nel riconoscimento di vari oggetti.
Per il riconoscimento di pedoni, metodi basati su Aggregated Channel Features sono quelli che invece garantiscono risultati migliori in termini di tempo di elaborazione ed accuracy.

Questo lavoro ha come obiettivo quello dell'addestramento di una Rete Neurale Convolutiva per il riconoscimento di pedoni, partendo da quello che è stato fatto, in modo da verificare se, come è accaduto nel riconoscimento di altri oggetti, l'utilizzo di CNN porta a risultati con un accuratezza almeno comparabile con i metodi allo stato dell'arte.

I risultati trovati confermano le aspettative: CNN ha successo nell'operazione di riconoscimento di persone, con prestazioni comparabili con quello di metodi allo stato dell'arte. Applicazioni come quelle per scopi militari, di sorveglianza o di tracciamento di pedoni per sistemi di assistenza su veicoli [8], [9], [10] possono trarre particolare beneficio dall'utilizzo da questi modelli, con il potenziale per salvare numerose vite.

Abstract

In the field of Artificial Intelligence, one of the latest goal that has recently been tried to achieve is the creation of systems with better "primary senses" than humans, or at least similar, like hearing and sight.

The state of the art in visual recognition is represented by Convolutional Neural Networks based algorithms. In particular, they have been proved to produce much more accurate results than other methods in classification tasks on a variety of objects.
In the case of pedestrian detection instead, Aggregated Channel Features achieves the best results in terms of both execution time and accuracy.

This work aims at training a Convolutional Neural Network for pedestrian detection, starting from what has been done so far to verify, like has happened detecting other objects, if CNNs produce results that are at least comparable with the state of the art ones.

The results confirm the trend: CNNs succeed in the task of pedestrian detection, with comparable performance to the state of the art methods. Applications such as military surveillance or automotive applications [8], [9], [10] are particularly compelling as they have the potential to save numerous lives.

Acknowledgements

I would like to express my gratitude to my advisor Prof. Marco Tagliasacchi for his willingness and for the support offered during the preparation of my thesis. This has been a very challenging yet incredible experience and I am really happy for the opportunity of working on this new topic which is defining the new frontier of computer vision.

I would like also to thank my assistant advisors, Luca Bondi and Luca Baroffio, for helping me to solve problems I faced with throughout all the thesis, supporting me and teaching me how to face problems in the research world.

My sincere thanks goes to Federico Monti for the mutual collaboration in our theses that led us at developing a method that compete with the state of the art ones.

Lastly, I would like to thank my family who always supported me and believed in me, making all of this possible.

Introduction

In the field of computer vision, the capability of reliably detect pedestrians in real-world images is interesting for several applications, such as

- intelligent video surveillance systems: it provides the fundamental information for semantic understanding of the video footage
- automatic driver-assistance systems in vehicles: due to the potential for improving safety systems
- part of driverless vehicle navigation systems.

At the same time, pedestrian detection is one of the most challenging categories for object detection, for a variety of reasons:

- Various style of clothing in appearance
- Different possible articulations
- The presence of occluding accessories
- Frequent occlusion between pedestrians

Finally, in many applications several people may be present in the same image region, partially occluding each other and adding to the difficulty.
Despite the challenges, pedestrian detection still remains an active research area in computer vision in recent years.

Many pedestrian detectors have been developed to address these challenges, such as holistic detection, e.g. the image features inside the local search window meet certain criteria [11] or histogram of oriented gradients [1]. Others use part-based detection, e.g. pedestrians are modeled as collections of parts [12],[13],[14], silhouettes [15],[16],[17]; or an assembly of local feature [18], and then employ classifiers such as boosting [19], and SVM [1] to decide wether a detected window should be classified as a pedestrian or not.

In recent years, deep learning has been applied to pedestrian detection and achieved promising results [20], [21]. Instead of using handcrafted features, it can automatically learn features in an unsupervised or supervised fashion. They are stacked into multiple layers so as to map the raw data into gradually higher-level representations [20]. The entire network is fine-tuned with label information and the top layer output is often adopted as features to train classifiers.

The growing number of low-power pervasive computing platforms available nowadays in industrial and scientific applications enforces the development of distributed intelligence Visual Sensor Networks. In particular, a basic Visual Sensor Network consists of a group of nodes, each equipped with a low power embedded processor, an energy source, one or more image sensors and a network adapter for communications [22]. These low power sensor nodes send data to a powerful central node, the sink, which takes care of collecting and processing all the data coming from the sensor nodes or even from other VSNs, in the workflow known as *Compress-Then-Analyze*.
Conversely the *Analyze-Then-Compress* paradigm [23] proposes another working schema: features extraction is performed directly on sensor nodes and a compact representation of the features is sent to the sink node. The ATC paradigm also reduces the load at the sink node, allowing more sensor nodes to send data to a single sink node sharing a limited bandwidth communication channel.

Neural networks are inherently parallel algorithms [24] that can be used with both *ATC* and *CTA* paradigms

- CTA: The node could be just the camera that compresses the video footage, and the sink node is a very powerful GPU that can quickly process all the incoming data from the different nodes.

- ATC: The node could be a dedicated hardware that acquires the content, processes it resorting to the CNN model, compresses the data and transmit them to the central controller that just interpret the information coming from the different nodes.

Ad-hoc hardware specifically designed on the neural network model, can take advantage of this parallelism, making fast parallel computation with a limited computational power.

The present work shows the sequence of steps performed for optimizing the accuracy of Convolutional Neural Network for pedestrian detection. Starting from a state-of-the-art method called R-CNN, this is extended by substituting some of its modules with other techniques that lead to better results. Using this, then some operations like filtering, tuning the model, ecc. are performed for a further result improvement.

Chapter 1 recaps the State of the Art for Pedestrian Detection analyzing all the methods, with particular interest in those that are extended by the proposed method.

Chapter 2 recaps the pedestrian datasets, showing which are the most used and the statistics of those datasets.

Chapter 3 analyzes the R-CNN method that has been used as a starting point, defining what we need to better perform in term of time and accuracy.

Chapter 4 defines a sliding window approach to prove the observations done in the previous chapter.

Chapter 5 defines an extended version of the sliding window approach that achieves state-of-the-art results.

Chapter 6 shows how the defined method has to be transformed in order to be able to apply the ATC paradigm on a specific hardware.

Finally, in Chapter 7 possible future works are proposed to continue the investigation on model compression, without loosing to much accuracy.

Chapter 1

Review of the State of the Art

In this section the state-of-the-art methods for pedestrian detection are analyzed. These are the milestones which defined the path for the presented work: from the first detection framework (Viola–Jones), moving to a very robust and widely used method (HOG), which has been integrated into the more recent ACF and LDCF methods, to finally consider the starting point of the thesis (AlexNet and R-CNN).

1.1 Viola – Jones detector

The Viola – Jones object detection framework is the first **object detection framework** to provide competitive object detection rates in real-time, proposed in 2001 by Paul Viola and Michael Jones [25]. Although it can be trained to detect a variety of object classes, it was motivated primarily by the problem of face detection.

The main characteristics of Viola–Jones algorithm which makes it a good detection algorithm are:

- Robust: very high true-positive rate and very low false-positive rate
- Real time
- Face detection and not recognition: distinguish faces from non-faces

The algorithm has mainly three stages:

1. Haar Features Selection
2. Creating Integral Image
3. Adaboost Classifier

The features employed by the detection framework universally involve the sums of image pixels within rectangular areas. Figure 1.1 illustrates the different types of features used in the framework. The value of any given feature is always simply the sum of the pixels within clear rectangles subtracted from the sum of the pixels within shaded rectangles. They are sensitive to vertical and horizontal features, and their feedback is considerably coarser.

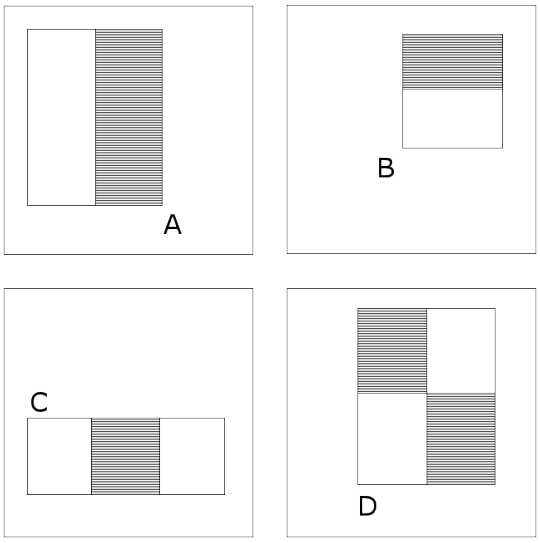
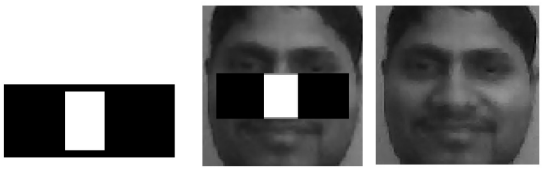


FIGURE 1.1: Viola – Jones feature types

All human faces share some similar properties. This knowledge is used to construct certain features known as **Haar Features**. The properties that are similar for a human face are:

- The eyes region is darker than the upper-cheeks
- The nose bridge region is brighter than the eyes



Haar Feature that looks similar to the bridge of the nose is applied onto the face

Haar Feature that looks similar to the eye region which is darker than the upper cheeks is applied onto a face

3rd and 4th kind of Haar Feature

FIGURE 1.2: Viola – Jones: Haar Features applied onto a face.

Knowing this, one can apply what is show in Figure 1.2.

To speed up the overall process, then with the use of an image representation called the **integral image**, rectangular features can be evaluated in constant time, which gives them a considerable speed advantage.

Finally, the identified features are input to the classifier.

1.2 Histogram of Oriented Gradients

The Histogram of Oriented Gradients is a feature descriptor used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in localized portions of an image.

The essential thought behind the HOG descriptor is that local object appearance and shape within an image can be described by the distribution of intensity gradients and

edge directions. The image is divided in small connected regions called **cells** (see Figure 1.3), and for the pixels within each cell, a HOG detection is compiled. The descriptor is then the concatenation of these histograms. For improved accuracy, the local histograms can be contrast-normalized by calculating a measure of the intensity across a larger region of the image, called a **block** (see Figure 1.3), and then using this value to normalize all cells within a block. This normalization results in better invariance to changes in illumination and shadowing. The overview of feature extraction and object detection is showed in Figure 1.4.

FIGURE 1.3: Blocks and Cells in HOG

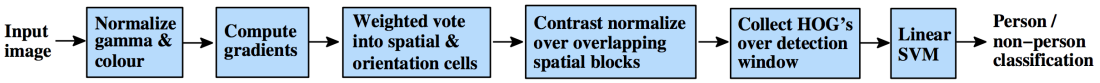


FIGURE 1.4: Overview of feature extraction and object detection in HOG

The first step consists in the computation of the gradient values. This is done by applying the 1-D centered, point discrete derivative mask in the horizontal and vertical directions. Specifically, it requires filtering the color or intensity data of the image with the following filter kernels:

$$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \qquad \text{and} \qquad \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}^T \tag{1.1}$$

Fox example, in Figure 1.5 is represented a subset of a grayscale image, showing the pixel values neighbouring to a specific point. Using both filters defined in Eq. 1.1, it is possible to compute the gradient of that specific point, as

$$\Delta f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} -1 * 56 + 1 * 94 \\ -1 * 93 + 1 * 55 \end{bmatrix} = \begin{bmatrix} 38 \\ -38 \end{bmatrix} \tag{1.2}$$

from which is then possible to extract the magnitude and the orientation (in the range $[0, 180]$)

$$\text{magnitude} = \sqrt{(-38)^2 + (38)^2)} = 53.74 \tag{1.3}$$

$$\text{orientation} = \arctan\left(\frac{\partial f}{\partial y}, \frac{\partial f}{\partial x}\right) = \arctan\left(\frac{38}{38}\right) = 45 \tag{1.4}$$

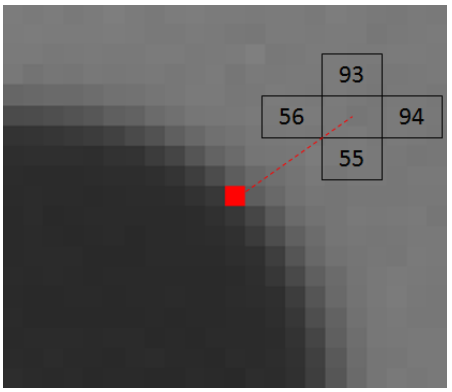


FIGURE 1.5: Filtering example

The second step is creating the cell histogram. Each pixel within the cell casts a weighted vote for an orientation based histogram channels based on the values found in the gradient computation. The histogram channels are evenly spread over 0 to 180 degrees.

As previously said, to account for changes in illumination and contrast, the gradient strength must be locally normalized, grouping cells into blocks. The HOG descriptor is then the concatenated vector of the components of the normalized cell histograms from all of the block regions.

For pedestrian detection, Dalal *et al.* [1] tested different configurations (see Figure 1.6) in order to find the one that produces the best results, which is:

- 9 orientation bins in the [0-180] orientation range
- 16×16 pixel blocks
- 8×8 pixel cells

- block spacing stride of 8 pixels: number of pixels the block is moved from one normalization to the next one (size of one cell), as visible in Figure 1.3.
- 64×128 detection window
- linear SVM classifier

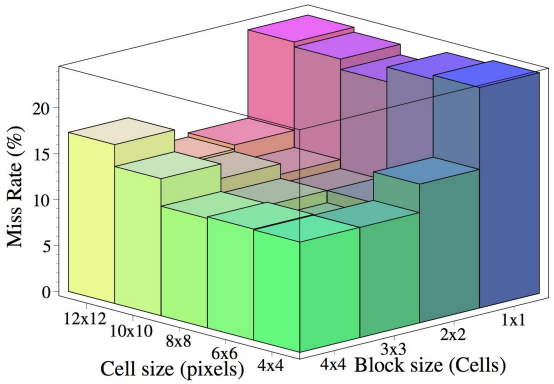


FIGURE 1.6: HOG block and cell size [1]

The HOG features overlapped to the image they are taken from, is shown in Figure 1.7.

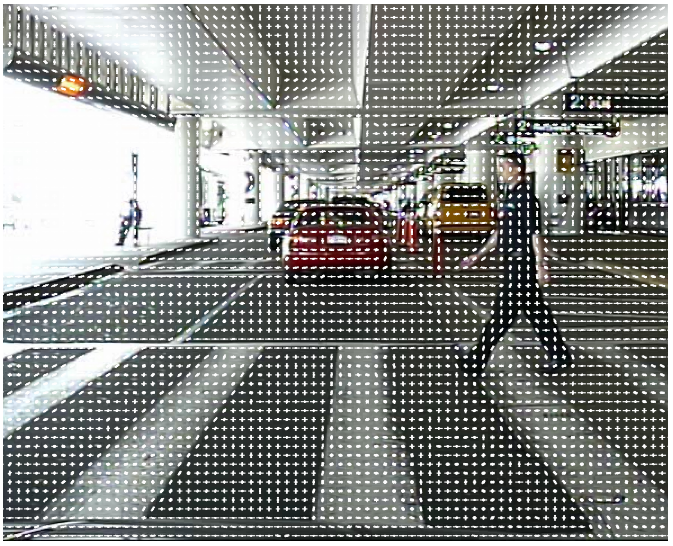


FIGURE 1.7: HOG features ovelapping the original image

1.3 Aggregated Channel Features

Aggregated Channel Features is an object detector method, based on Fast Features Pyramids [2]. It proves that the behaviour of image features can be predicted reliably across scales, showing that is possible to estimate features at a given scale inexpensively, by extrapolating computation carried out at a coarsely sampled set of scales.

1.3.1 Fast Features Pyramids

The idea behind Fast Features Pyramids is that intuitively, the information content of an upsampled image is similar to that of the original, lower resolution image. In particular, let $M'(i,j) \approx \frac{1}{k} M\left(\lceil \frac{i}{k} \rceil, \lceil \frac{j}{k} \rceil\right)$ denote the *gradient magnitude* in an upsampled discrete image. Then

$$\sum_{i=1}^{kn} \sum_{j=1}^{km} M'(i,j) \approx k \sum_{i=1}^{n} \sum_{j=1}^{m} M(i,j) \tag{1.5}$$

Thus, the sum of gradient magnitudes in the original and upsampled image should be related by a factor of k. Therefore, according to the definition of gradient histograms, what happens is that: $h'_q \approx k h_q$, allowing us to approximate gradient histograms in an upsampled image using gradients computed at the original scale.

Considering down-sampled images, Piotr *at al.* [2] proved that Eq. 1.5 applies as well. An example of the application can be found in Figure 1.8

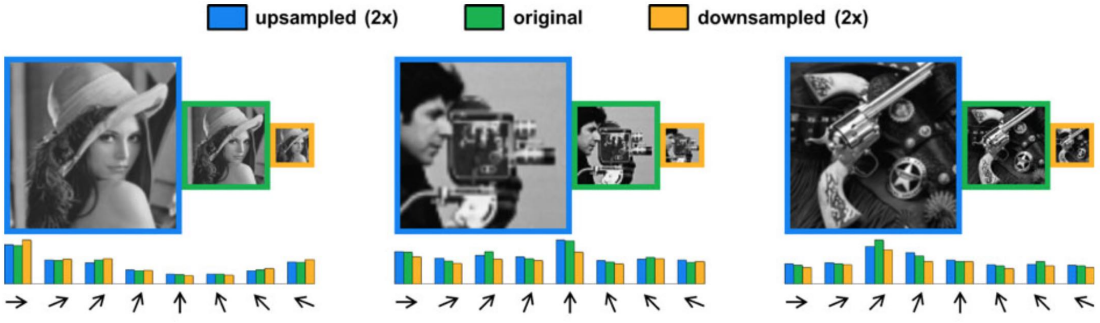


FIGURE 1.8: Approximating gradient histograms in images resampled by a factor of two [2]

An important formula governing feature scaling, is the *power law*,

$$\frac{f_\Omega(I_{s_1})}{f_\Omega(I_{s_2})} = (s_1/s_2)^{-\lambda_\Omega} + \varepsilon \tag{1.6}$$

where

- I_s represents image I at scale s
- $f_\Omega(I)$ denotes an arbitrary scalar image statistic
- λ_Ω represents the coefficient for which the power law is satisfied for a statistic
- ε denotes the deviation from the power law for a given image

and each image statistic has its own corresponding λ_Ω, which is possible to determine empirically.

Considering a general image, a "channel" is a conventional term used to refer to a certain component of that image. The simplest example of channels are the three components defining any coloured image: R, G and B (Red, Green and Blue). Staring from the power law governing feature scaling (Eq. 1.6), an extension that applies directly to channel images is proposed: the standard approach to compute features at different scales, was to compute the channel at scale s, $C_s = \Omega(R(I, s))$, where $R(I, s)$ denotes image I resampled by s, without considering the information contained in the original scale image, $C = \Omega(I)$. Conversely, the new approach (see Figure 1.9) proposes the following approximation, using the already available data

$$C_s \approx R(C, s) \cdot s^{-\lambda_\Omega} \tag{1.7}$$

Feature pyramid is the multi-scale representation of an image I where channels are computed at every scale s. What is done is basically compute $C_s = \Omega(R(I, s))$ at just one scale per octave $\left(s \in \{1, \frac{1}{2}, \frac{1}{4}, \ldots\}\right)$, while at intermediate scales, C_s is computed using Eq. 1.7 (see Figure 1.10). This provides a good tradeoff between speed and accuracy.

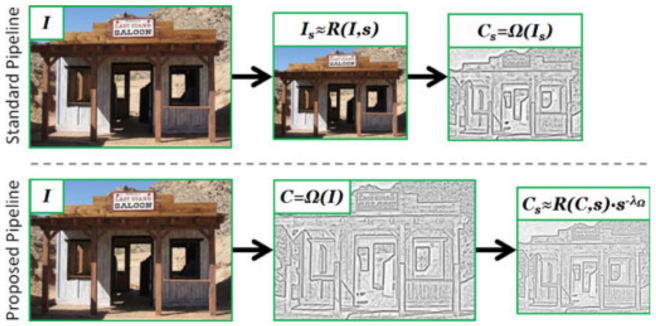


FIGURE 1.9: Feature channel scaling [2]

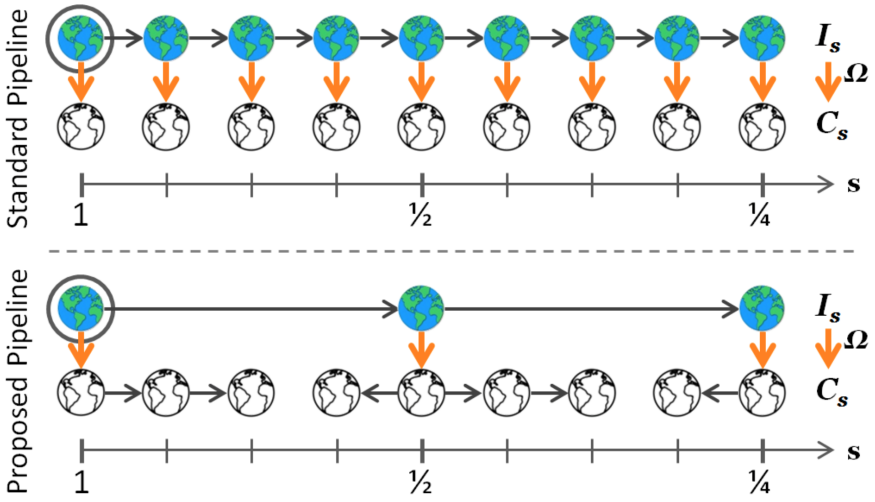


FIGURE 1.10: Fast feature pyramids. Color and grayscale icons represent images and channels [2]

1.3.2 ACF

The ACF detection framework is conceptually straightforward (see Figure 1.11). Given an image I, compute several intermediate channels $C = \Omega(I)$, sum every block of pixels in C, and finally smooth the resulting lower resolution channel.

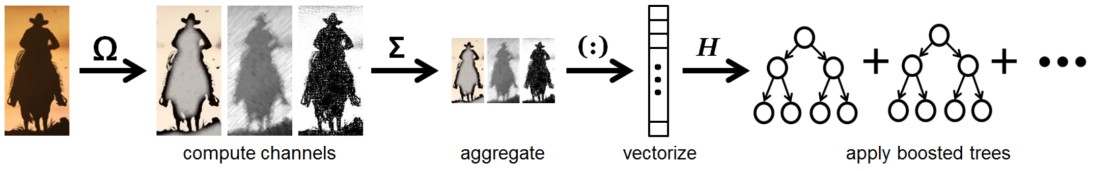
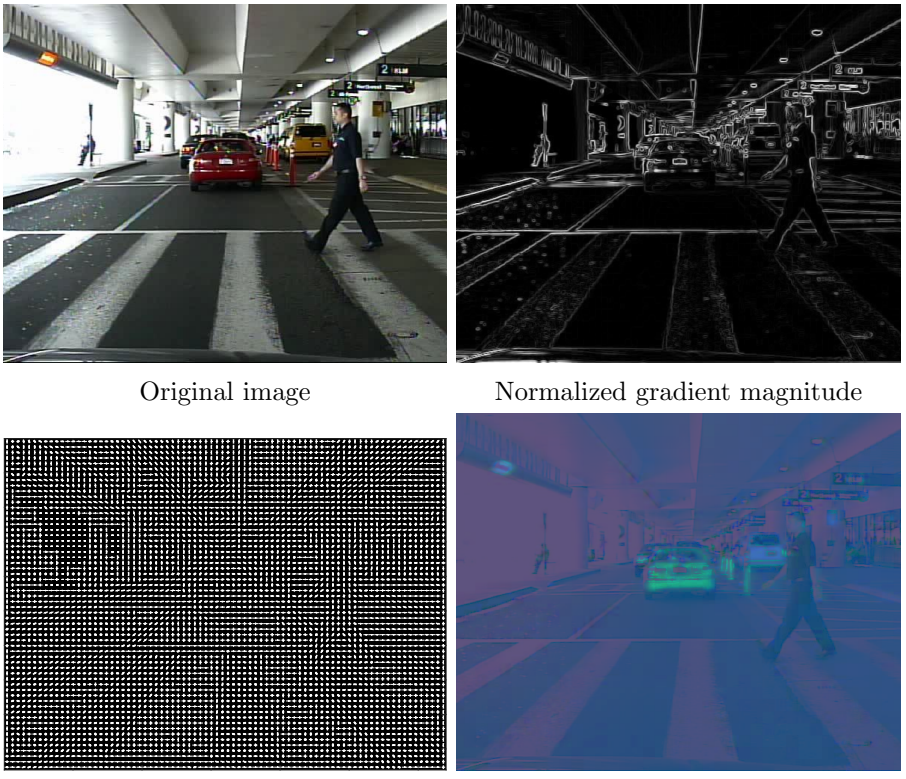


FIGURE 1.11: Overview of the ACF detector. Boosting is used to learn decision trees over these features (pixels) to distinguish object from background.

The channels used by ACF are

- normalized gradient magnitude
- histogram of oriented gradients (see Sec. 1.2)
- LUV color channels

These channels are show in Figure 1.12, computed on one frame taken from the Caltech pedestrian dataset.



Original image | Normalized gradient magnitude

Histogram of oriented gradients | LUV color channels

FIGURE 1.12: ACF channels computed on an image

In the channels computation, Fast Features Pyramids (see Sec. 1.3.1) is used to compute the approximations at octave-spaced scale intervals. After that, a sliding window approach is used. This is a very simple method where a window of size 128×64 is moved over the image, defining which portion of image has to be considered in our computation for extracting the information we need.

For pedestrian detection, AdaBoost [26] classifier is used to train and combine a lot of depth-two trees over the candidate features in each 128×64 window. For 640×480 images, the complete system, including fast pyramid construction and sliding-window detection, runs at over 30 fps.

1.4 Locally Decorrelated Channel Features

Decision trees with orthogonal splits – also known as axis-aligned decision trees – are very popular in detection. They work by deciding at each decision node, the single feature to be used, that best splits the data. A possible explanation for the persistence of orthogonal splits is their efficiency: oblique (multiple feature) splits incur considerable computational cost during both training and detection. Nevertheless, oblique trees can hold considerable advantages.

To achieve similar advantages of oblique trees, LDCF [27] proposes to decorrelate features prior to applying orthogonal trees. To do so, it introduces an efficient feature transform that removes correlations in local image neighborhoods. The result is an overcomplete but locally decorrelated representation that is ideally suited for use with orthogonal trees.

In oblique trees, every split is based on a linear projection of the data $\mathbf{z} = \mathbf{w^T x}$, followed by thresholding. To obtain w in practice, linear discriminant analysis (LDA) is a natural choice for obtaining discriminative splits efficiently [28]. LDA aims to minimize within-class scatter while maximizing between-class scatter. In particular

$$w = \Sigma^{-1} (\mu_+ - \mu_-) \tag{1.8}$$

where Σ is a class-independent covariance matrix. Let $\Sigma = Q \Lambda Q^T$ be the eigendecomposition of Σ, then by this, the following transformations are defined:

- *decorrelation*: Q^T
- *PCA-whitening*: $\Lambda^{-\frac{1}{2}} Q^T$

- *ZCA-whitening*: $Q\Lambda^{-\frac{1}{2}}Q^T$

Figure 1.13 shows the result of boosting *orhtogonal* decision trees on the various transformed features. Therefore, it's possible to create a decorrelated representation by computing $\mathbf{Q^T p}$.

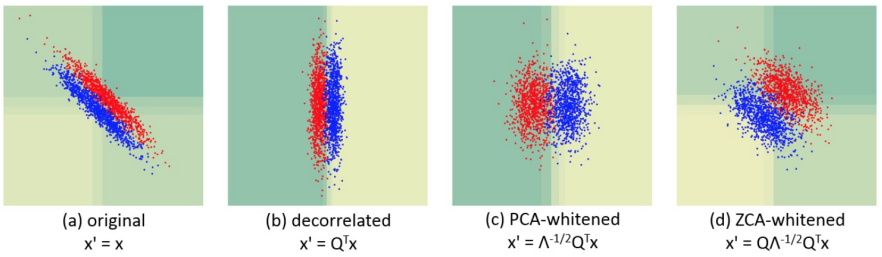


FIGURE 1.13: A comparison of boosting with orthogonal decision trees on transformed data. Orthogonal trees with both decorrelation and PCA-whitened features show improved generalization, while ZCA-whitening is ineffective.

To reduce dimensionality and to speed up the orthogonalization process, LDCF proposes to utilize the top-k eigenvectors in Q: the intuition is that the top eigenvectors capture the salient neighborhood structure (experimentally confirmed [28]).

Given the new locally decorrelated channels, all other training and testing ACF's steps remain identical (see Sec. 1.3).

1.5 Convolutional Neural Networks

Deep learning is a branch of machine learning that is advancing the state of the art for perceptual problems like vision and speech recognition. We can pose these tasks as mapping concrete inputs such as image pixels or audio waveforms to abstract outputs like the identity of a face or a spoken word. The "depth" of deep learning models comes from composing functions into a series of transformations from input, through intermediate representations, and onto output. The overall composition gives a deep, layered model, in which each layer encodes progress from low-level details to high-level concepts. This yields a rich, hierarchical representation of the perceptual problem.

The strength of deep models is that they are not only powerful but learnable. The capacity to represent a function is not enough if all the details of it cannot be described and engineered. The visual world is too vast and varied to fully describe by hand, so it has to be learned from data. A deep net is trained by feeding it input and letting it compute layer-by-layer to generate output for comparison with the correct answer.

Convolutional Neural Networks (CNN) are a particular type of deep models responsible for many exciting recent results in computer vision, *e.g.* **AlexNet** [4] in 2012, which won the world-wide ImageNet Large-scale Visual Recognition Challenge (ILSVRC).

In a CNN, the key computation is the convolution of a feature detector with an input signal (see Figure 1.14). At the first layer of a CNN the features go from individual pixels to simple primitives like horizontal and vertical lines, circles, and patches of color. In contrast to conventional single-channel image processing filters, these CNN filters (translation-invariant) are computed across all of the input channels.

Figure 1.15 shows what the network has learned by computing its top-5 predictions on eight test images.

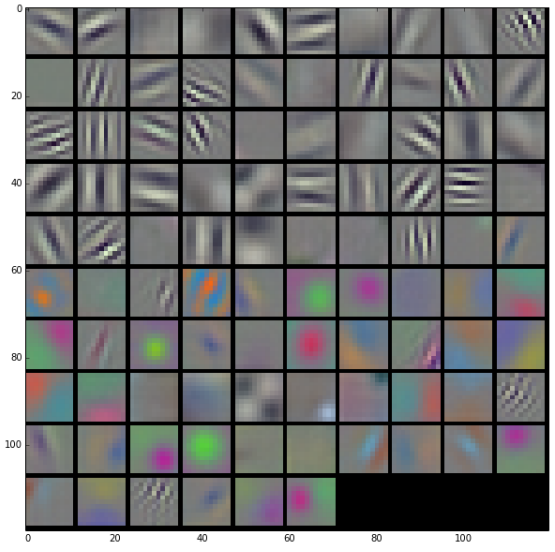


FIGURE 1.14: First layer of the learned convolutional filters [3]

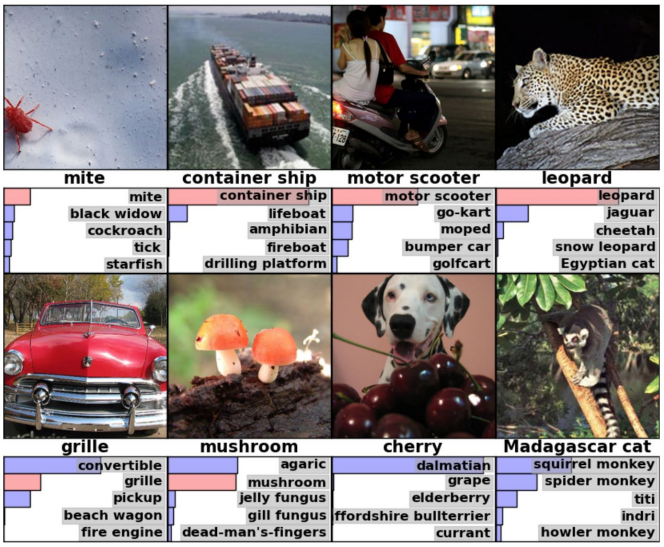


FIGURE 1.15: Test images and the five labels considered most probable by this model

1.5.1 Architecture

As shown in Figure 1.16, the net contains eight layers with weights; the first five are convolutional and the remaining three are fully-connected, which means that every input of the layers, is taken into account to compute every output of the same layers.

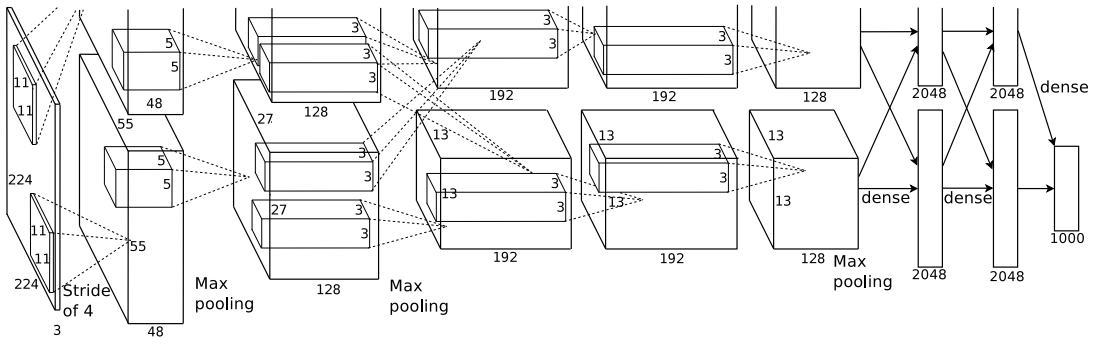


FIGURE 1.16: Illustration of the architecture of the CNN [4]

Convolutional layers consist of a rectangular grid of neurons. They require the previous layer also to be a rectangular grid of neurons. Each neuron takes inputs from a rectangular section of the previous layer; there are many convolution kernels in each layer, and each kernel is replicated over the entire image with the same parameters.

Thus, the convolutional layer is just an image convolution of the previous layer, where the weights specify the convolution filter and they are taken by a **convolutional kernel**.

Suppose there is some $N \times N$ square neuron layer which is followed by a convolutional layer. If an $m \times m$ filter ω is used, the convolutional layer output will be of size $(N - m + 1) \times (N - m + 1)$. In order to compute the pre-nonlinearity input to some unit x_{ij}^l in the layer, we need to sum up the contributions (weighted by the filter components) from the previous layer cells:

$$x_{ij}^l = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \omega_{ab} \cdot y_{(i+a)(j+b)}^{l-1} \tag{1.9}$$

In Figure 1.17 an example of convolution.

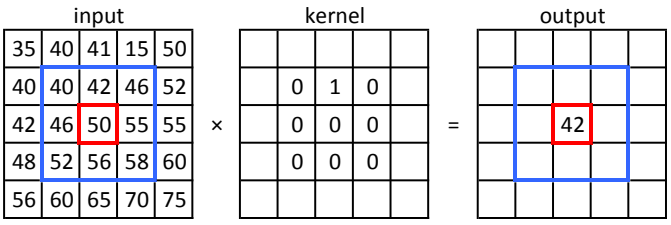


FIGURE 1.17: Convolution layer: example

Then, the convolutional layer applies its nonlinearity:

$$y_{ij}^l = h(x_{ij}^l) \tag{1.10}$$

Unlike a hand-coded convolution kernel (Sobel, Prewitt, Roberts), in a CNN, the parameters of each convolution kernel are trained by the backpropagation algorithm. The function of the convolution operators is to extract different features of the input (see Figure 1.18).

Pooling layers in CNNs summarize the outputs of neighboring groups of neurons in the same kernel map. Traditionally, the neighborhoods summarized by adjacent pooling units do not overlap. To be more precise, a pooling layer can be thought of as consisting of a grid of pooling units spaced s pixels apart, each summarizing a neighborhood of size $z \times z$, centred at the location of the pooling unit. By setting $s = z$, we obtain traditional local pooling as commonly employed in CNNs. By setting $s < z$, we obtain overlapping

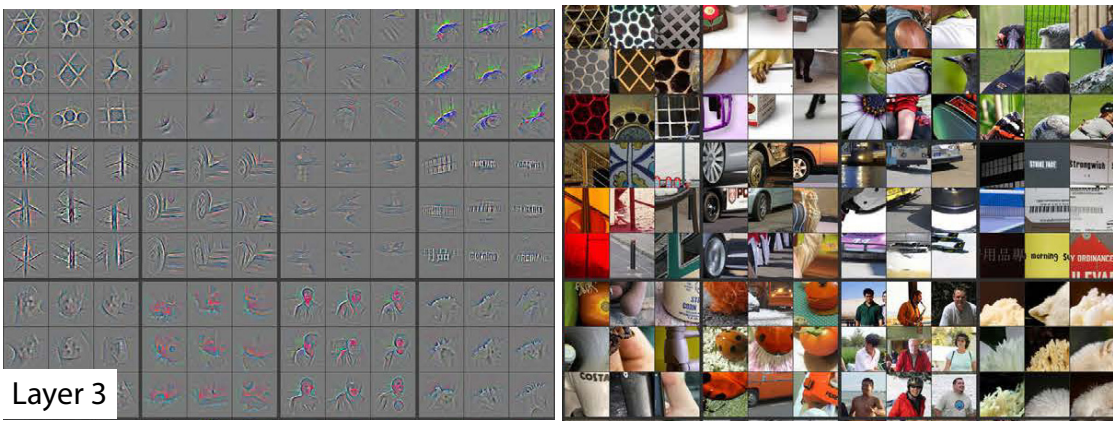


FIGURE 1.18: AlexNet: visualization of features in a fully trained model, showing the top 9 activations, projected down to pixels using deconvolutional network [5] approach.

pooling. This is what it's use throughout this network, with $s = 2$ and $z = 3$ and an example is shown in Figure 1.19.

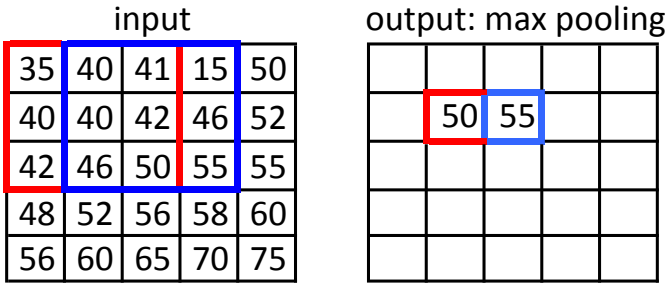


FIGURE 1.19: Max pooling layer: example

This scheme reduces the top-1 and top-5 error rates by 0.4% and 0.3%, respectively, as compared with the non-overlapping scheme $s = 2$, $z = 2$, which produces output of equivalent dimensions [4].

Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via **fully connected layers**. A fully connected layer takes all neurons in the previous layer (be it fully connected, pooling, or convolutional) and connects it to every single neuron it has. Fully connected layers are not spatially located anymore, so there can be no convolutional layers after a fully connected layer.

The output of the last fully-connected layer is fed to a 1000-way softmax which produces a distribution over the 1000 class labels.

1.5.2 Back propagation

Back propagation is a common method for training artificial neural networks used in conjunction with an optimization method such as gradient descent. The method calculates the gradient of a loss function with respect to all the weights in the network. The gradient is fed to the optimization method which in turn uses it to update the weights, in an attempt to minimize the loss function.

Suppose the training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$ of m training examples is given. For a single training example (x,y), a cost function with respect to that single example is defined to be

$$J(W, b; x, y) = \frac{1}{2} \left\| h_{W,b}(x) - y \right\|^2 . \tag{1.11}$$

where $h_{W,b}(x)$ is the resulting value after applying the activation function to $W \cdot x + b$. The overall cost function is then

$$J(W, b) = \left[\frac{1}{m} \sum_{i=1}^{m} J(W, b; x^{(i)}, y^{(i)}) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l - 1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{(l)} \right)^2 \tag{1.12}$$

$$= \left[\frac{1}{m} \sum_{i=1}^{m} \left(\frac{1}{2} \left\| h_{W,b}(x^{(i)}) - y^{(i)} \right\|^2 \right) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l - 1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{(l)} \right)^2 \tag{1.13}$$

where n_l is the number of layers and s_l the number of neurons in the l-th layer.

The first part of the formula is an average of sum-of-square errors while the second part is the weight decay term, used for decreasing the magnitude of the weights, and helps prevent overfitting: the **weight decay parameter** λ controls the relative importance of the two terms.

To train the neural network, each parameter $W_{ij}^{(l)}$ and $b_i^{(l)}$ is initialized to a small random value near zero according to a Normal$(0, \varepsilon)$ distribution for some small ε, and then apply an optimization algorithm such as **batch gradient descent**.

The Gradient descent update formula for both W,b parameters is as follows:

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) \tag{1.14}$$

$$b_{i}^{(l)} = b_{i}^{(l)} - \alpha \frac{\partial}{\partial b_{i}^{(l)}} J(W,b) \tag{1.15}$$

where α is the **learning rate**. In this way, at each iteration, all the weights contained in the model are updated layer-by-layer, considering the error that has been done with the pervious weight values.

Backpropagation provides an efficient way for computing the derivatives, and the intuition behind it is as follows. Given a training example (x,y), "forward pass" is run to compute all the activations throughout the network, including the output value of the hypothesis $h_{W,b}(x)$. Then, for each node i in layer l, an "error term" δ_i^l is computed. It measures how much a node is "responsible" for any errors in the output. For an output node, we can directly measure the difference between the network's activation and the true target value, and use that to define $\delta_i^{n_l}$ (where layer n_l is the output layer).

In details:

1. For each output i in layer n_l (the output layer), set

$$\delta_i^{(n_l)} = \frac{\partial}{\partial W_{ij}^{(n_l)}} \ \frac{1}{2} \left\| y - h_{W,b}(x) \right\|^2 = -(y_i - h(a_i^{(n_l)})) \cdot h'(a_i^{(n_l)}) \tag{1.16}$$

   where $a_i^{(n_l)}$ is the weighted sum and h the activation function.

2. For $l = n_l - 1, \cdots, 2$ and for each node i in layer l, set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) h'(a_i^{(l)}) \tag{1.17}$$

3. Compute the desired partial derivatives, which are given as

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)} \tag{1.18}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y) = \delta_i^{(l+1)}. \tag{1.19}$$

4. Compute the derivatives of the overall cost function

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) = \left[\frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x^{(i)},y^{(i)}) \right] + \lambda W_{ij}^{(l)} \tag{1.20}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W,b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial b_i^{(l)}} J(W,b;x^{(i)},y^{(i)}) \tag{1.21}$$

Here weight decay is applied only to W but not to b.

1.5.3 Dropout

The recently-introduced technique, called "dropout" [29], consists of setting to zero the output of each hidden neuron with probability 0.5. The neurons which are "dropped out" in this way do not contribute to the forward pass and do not participate in back-propagation. So every time an input is presented, the neural network samples a different architecture, but all these architectures share weights. This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.

1.6 R-CNN

Regions with CNN features (R-CNN) [30] object detection system consists of three modules:

- The first generates **category-independent** region proposals. These proposals define the set of candidate detections available to our detector.

- The second is a large CNN (e.g. AlexNet, see Sec. 1.5) that extract a fixed length feature vector from each region.
- The third module is a set of class-specific linear SVMs.

1.6.1 Region Proposal

A variety of recent papers offer methods for generating *category-independent* region proposals. Among these, Selective Search [31] has been chosen. This uses a data-driven grouping-based strategy, where it increases diversity by using a variety of complementary grouping criteria and a variety of complementary colour spaces with different invariance properties. The goal is to generate a class-independent, data-driven, selective search strategy that generates a small set of high-quality object locations.

A selective search algorithm is subject to the following design considerations

- **Capture All Scales.** Object can occur at any scale within the image. Furthermore, some objects have less clear boundaries then other objects.
- **Diversification.** There is no single optimal strategy to group regions together.
- **Fast to compute.** The goal of selective search is to yield a set of possible object locations for use in a particular object recognition framework.

Bottom-up grouping is a popular approach to segmentation, hence this is adapted for selective search. Because the process of grouping itself is hierarchical, we can naturally generate locations at all scales by continuing the grouping process until the whole image becomes a single region.

To create initial regions a method defined by Felzenszwalb *et al.* [32] is used. It defines some initial beans placed on the image, which are then left to grow using a greedy algorithm that iteratively group regions together. First the similarities between all neighbouring regions are calculated. The two most similar regions are grouped together, and new similarities are calculated between the resulting region and its neighbours. The

process of grouping the most similar regions is repeated until the whole image becomes a single region.

For the similarity $s(r_i, r_j)$ between region r_i and r_j a variety of complementary measures is used, under the constraint that they are fast to compute. In particular these complementary similarity measures are

- **colour similarity**

$$s_{\text{colour}}(r_i, r_j) = \sum_{k=1}^{n} \min(c_i^k, c_j^k) \tag{1.22}$$

- **texture similarity**

$$s_{\text{texture}}(r_i, r_j) = \sum_{k=1}^{n} \min(t_i^k, t_j^k) \tag{1.23}$$

- **size similarity**

$$s_{\text{size}}(r_i, r_j) = 1 - \frac{\text{size}(r_i) + \text{size}(r_j)}{\text{size}(im)} \tag{1.24}$$

- **fill similarity**

$$s_{\text{fill}}(r_i, r_j) = 1 - \frac{\text{size}(BB_{ij}) - \text{size}(r_i) - \text{size}(r_j)}{\text{size}(im)} \tag{1.25}$$

The final similarity measure is a combination of the above four

$$s(r_i, r_j) = a_1 s_{\text{colour}}(r_i, r_j) + a_2 s_{\text{texture}}(r_i, r_j) + a_3 s_{\text{size}}(r_i, r_j) + a_4 s_{\text{fill}}(r_i, r_j) \tag{1.26}$$

1.6.2 Feature extraction

A 4096-dimensional feature vector is extracted from each region proposal using the implementation of the CNN described by Krizhevsky *et al.* [4]. Features are computed by forward propagating a mean-subtracted 227×227 RGB image through five convolutional layers and two fully connected layers (not three because the last two layers are substituted by a SVM classifier). Since this architecture requires inputs of a fixed 227×227 pixel size, regardless of the size or aspect ratio if the candidate region, all pixels are warped in a tight bounding box around it to the required size.

Detection average precision (%) on VOC 2010 test can be seen in Table 1.1.

VOC 2010 test	aereo	bike	bird	boat	bottle	bus	car	cat	chair	cow	
R-CNN	67.1	64.1	46.7	32.0	30.5	56.4	57.2	65.9	27.0	47.3	
VOC 2010 test	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
R-CNN	40.9	66.6	57.8	65.9	53.6	26.7	56.5	38.1	52.8	50.2	50.2

TABLE 1.1: *Detection average precision* on VOC 2010 test.

Chapter 2

Datasets

2.1 Pascal Visual Object Classes

The PASCAL Visual Object Classes (VOC) challenge is a benchmark in visual object category recognition and detection, providing the vision and machine learning communities with a standard dataset of images and annotation, and standard evaluation procedures. Organised annually from 2005 to present, the challenge and its associated dataset has become accepted as the benchmark for object detection.

For the 2007 challenge, all images were collected from the Flickr photo-sharing website. The use of personal photos which were not taken by, or selected by, vision/machine learning researchers results in a very 'unbiased' dataset, in the sense that the photos are not taken with a particular purpose in mind i.e. object recognition research. Qualitatively the images contain a very wide range of viewing conditions (pose, lighting, etc.) and they are not focused on a particular object, e.g. there are images of motorcycles in a street scene, rather than solely images where a motorcycle is the focus of the picture.

In total, 500.000 images were retrieved from Flickr. For each of the 20 object classes to be annotated (see Figure 2.1), images were retrieved by querying Flickr with a number of related keywords.

FIGURE 2.1: Example of images from the VOC2007 dataset. Bounding boxes indicate all instances of the corresponding class in the image.

2.2 Caltech Dataset

The *Caltech Pedestrian Dataset* consists of approximately 10 hours of 640x480 30Hz video taken from a vehicle driving through regular traffic in an urban environment. About 250.000 frames (corresponding to $\sim$ 137 minutes) with a total of 350.000 bounding boxes and 2300 unique pedestrians were annotated. The annotation includes temporal correspondence between bounding boxes and detailed occlusion labels (bounding boxes that denote the visible and full pedestrian extent).

This dataset is two order of magnitude larger than any existing dataset. The pedestrians vary widely in appearance, pose and scale; furthermore, occlusion information is annotated (see Figure 2.2). These statistics are more representative of real world applications and allow for in depth analysis of existing algorithms.

FIGURE 2.2: Example images (cropped) and annotations. The solid green boxes denote the full pedestrian extent while the dashed yellow boxes denote the visible regions [6]

About 50% of the frames have no pedestrians, while 30% have two or more. Pedestrians are visible for 5s on average. In Figure 2.3, there is a detailed analysis of the distribution of pedestrian scale. This serves as a foundation for establishing the requirements for a real world system.

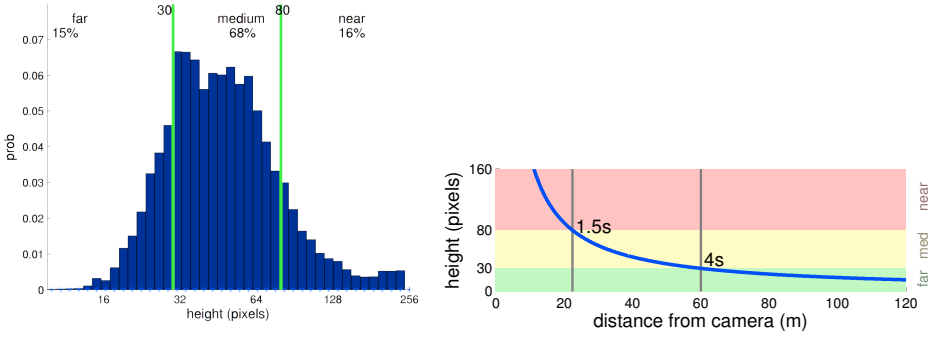


FIGURE 2.3: Pedestrian height distribution. Most pedestrians are observed at the medium scale (30-80 pixels)

2.2.1 Training and Testing Data

The Caltech Pedestrian Dataset was captured over 11 sessions. These sessions are divided as follows:

- Train-set: from set00 to set05
- Test-set: from set06 to set10

In particular, statistics on train/test sets can be found in table 2.1.

	Training			Testing		
	# pedestrians	# neg. images	# pos. images	# pedestrians	# neg. images	# pos. images
Caltech	192k	61k	67k	155k	56k	65k

TABLE 2.1: *Caltech statistics*

Furthermore, what has been proven to achieve better result [33] is to extract from train and test datasets images with a different sampling rate:

- train set: extract 1 image every 3 frames, e.g. 2, 5, ...
- test set: extract 1 image every 30 frames, e.g frame 29, 59, ...

This leads to a filtered dataset, whose main features are shown in Table 2.2.

	set	**num images**	**num positive regions**
train	0	8559	7232
	1	3619	2903
	2	7410	588
	3	7976	3023
	4	7328	1235
	5	7890	1394
test	6	1155	903
	7	746	1297
	8	657	352
	9	738	557
	10	728	776

TABLE 2.2: *Caltech dataset: number of images per set*

These data can be arranged in different scenarios for training the model:

- *Scenario-A*: Train using sessions 0–5, test on sessions 6–10
- *Scenario-B*: Perform 6-fold cross validation using train session, test on test sessions

This dataset has been tested with a variety of state of the art methods in pedestrian detection (see Figure 2.4), therefore using this dataset gives us a very clear idea about how our method performs in a real world scenario with respect to the other methods.

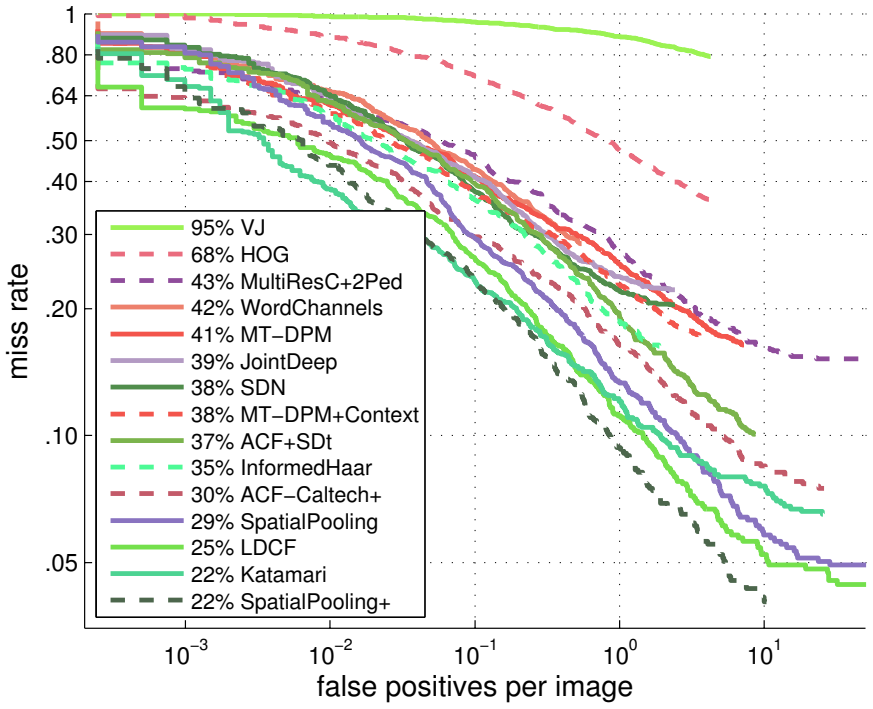


FIGURE 2.4: Performance on Caltech Pedestrian dataset on unoccluded pedestrians over 50 pixels tall

Chapter 3

R-CNN analysis

R-CNN method is a good starting point to achieve the desired goal. In order to understand how this method performs on the reference dataset, if and how it can be improved, an analysis has to be performed.

3.1 Analysis on region proposal selector

R-CNN, as described in sec. 1.6, is composed of three modules:

- Selective Search: region proposal selector
- CNN: computes features
- SVM: defines score for each region

The first one is essential in order to achieve good results (garbage-in-garbage-out): for instance, if many good regions are discarded, CNN can only score the remaining regions with low values; all the already filtered-out regions are lost and this results in a lot of possible misses.

By considering the Caltech Pedestrian Dataset (see sec. 2.2) we choose to test R-CNN in what is called by Piotr *at al.* [7] a **reasonable evaluation dataset**: evaluate

performance on pedestrian at least 50 pixels tall under no occlusion. This because medium scale detection is essential for automotive applications. In fact, the distance of a pedestrian can be computed using

$$h \approx H\frac{f}{d} \tag{3.1}$$

where H is the true object height, f the focal length and d the distance from the camera. Assuming $H \approx 1.8m$ tall pedestrians and having $f \approx 1000$ in pixels, we obtain $d \approx \frac{1800}{h}$ m. With the vehicle travelling at an urban speed of 55 km/h a 30 pixels person is 4 seconds away. We can exclude those pedestrian that are less relevant.

Given a set of detected and ground truth bounding boxes, using R-CNN or other methods, there is also the important task of understanding which and if a detected bounding box corresponds to a ground truth one. This is performed using the **Intersection over Union** metric, used by a greedy algorithm:

$$IoU = \frac{area(BB_{dt} \cap BB_{gt})}{area(BB_{dt} \cup BB_{gt})} \tag{3.2}$$

where

- BB_{dt} is the bounding box detected by the selector
- BB_{gt} is the bounding box ground truth, defined by the dataset

If $IoU > 0.5$ then two BBs i and j are considered representing the same person. The greedy algorithms takes then the list of BB_{dt}, sort them according to the score given by the detector and then tries to find a match with the BB_{gt}'s. Every time a match is found, the corresponding BB_{gt} is removed from the list.

Finally, the **Receiver Operating Characteristic** (ROC) curve is a graphical plot that illustrates the performance of a classifier as its discrimination threshold is varied. The ROC of R-CNN is shown in Figure 3.1, computed using the Matlab toolbox provided by *Piotr Dollar* on the reasonable evaluation dataset. Furthermore, it is computed using a portion of the test-set which is representative enough of the whole test-set. On the

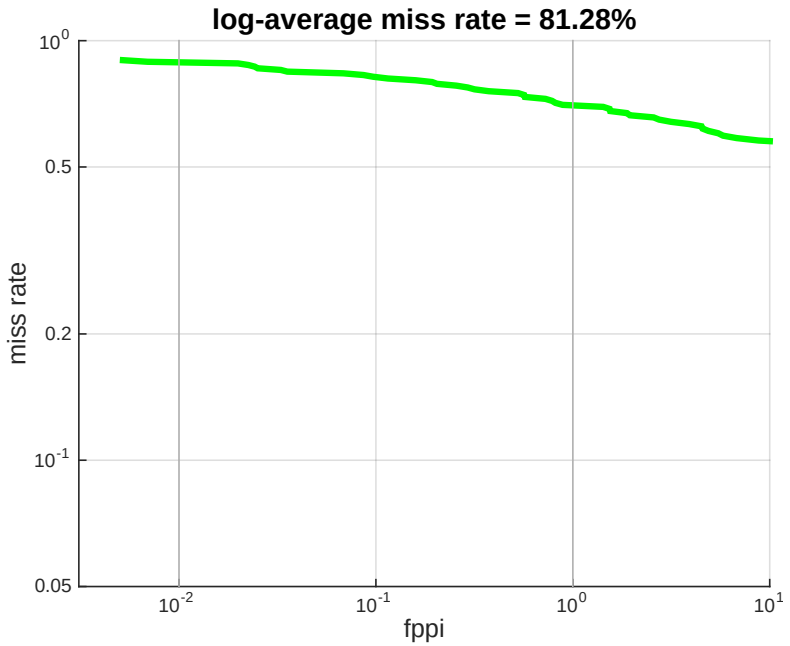


FIGURE 3.1: R-CNN ROC computed on set 06 of the test-set

horizontal axis is represented the false positive per image value:

$$fppi = \frac{\text{number false positives}}{\text{number of images}} \tag{3.3}$$

while on the vertical axis the miss rate, defined as

$$\text{miss rate} = \frac{np - tp}{np} = \frac{fn}{np} \tag{3.4}$$

where np is the number of ground truth regions, tp the number of correctly detected regions and fn the number of missed regions. This plot allows, in addition to performance comparison, also to identify which is threshold to be used on the scores, in order to have results in the working point.

Analyzing the figure, it's possible to see that if we want to have 0.1 false positive per image (fppi), then the relative miss rate generated by the model is around 0.8, meaning that every 10 images we consider, on average 1 object is wrongly classified as a person, missing 80% of the people.

The ROC curve also shows the log-average miss rate value, which is a geometric mean

$$\left(\prod_{i=1}^{n} a_i\right)^{1/n} = exp\left[\frac{1}{n}\sum_{i=1}^{n}\ln\left(a_i\right)\right] \tag{3.5}$$

computed using as data the miss rates sampled in $\begin{bmatrix}0.0100 & 0.0178 & 0.0316 & 0.0562 & 0.1000 & 0.1778 & 0.3162 & 0.5623 & 1.0000\end{bmatrix}$ fppi. It's just a method used to roughly represent the goodness of a model using a scalar value (useful for model performance comparison). The range $\begin{bmatrix}0.01 & 1\end{bmatrix}$ has been choosen according to [30].

As already mentioned, one important consideration about R-CNN is that the regions are proposed by Selective Search and scored by CNN, therefore if the first module is not able to propose the correct regions, the following one would just give low scores.

The first plot seems to confirm that Selective Search is not a good selector for our dataset, however to further inspect this observation, a test can be easily performed by plotting at the ROC plot extended to 10^3 fppi (see Figure 3.2).

Looking at 10^3 fppi, it's almost the same of looking at everything that Selective Search proposes: the threshold defined to be in that working point is so small that the score given by CNN is always bigger than that threshold. Furthermore, the best achievable result is 50% miss rate: even if the CNN model is trained to achieve its best result identifying correctly all the non-pedestrian proposals (working point close to 10^{-2} fppi), it still misses 50% of the people that were there.

The results with this particular method cannot be improved, because what is lost is lost, and all the "good" regions filtered out by Selective Search cannot be retrieved, therefore it's not possible to have better performance.

3.2 Result comparison

Figure 3.4 shows the ROC curve where R-CNN is compared with two important methods in the field of object detection

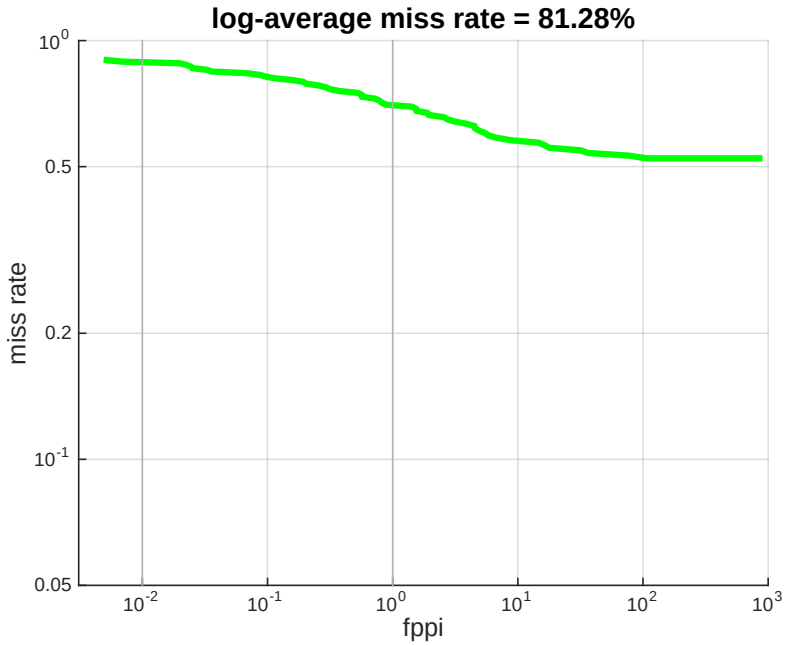


FIGURE 3.2: R-CNN ROC extended to 10^3 fppi computed on set 06 of the test-set

- HOG: Histogram of Oriented Gradients (see Sec. 1.2)
- LDCF: Locally Decorrelated Channel Features (see Sec. 1.4)

Both methods use the sliding window approach, and features of each region are scored using a classifier. They perform much better than R-CNN and the difference starts to be relevant from 10^0 fppi. It's important to mention that while R-CNN and HOG are trained on dataset different from the Caltech Pedestrian one (see sec. 2.2), LDCF is trained exactly on that one. This justifies the difference in performance among them.

Always considering the selector, to inspect more in depth the importance of the region proposals, an analysis of HOG and LDCF selectors is performed. Considering Figure 3.3 it's possible to see that HOG produces much more misses than LDCF, resulting in a ROC curve (see Figure 3.4) where acceptable results (0.2 miss rate) occurs with a too large fppi.

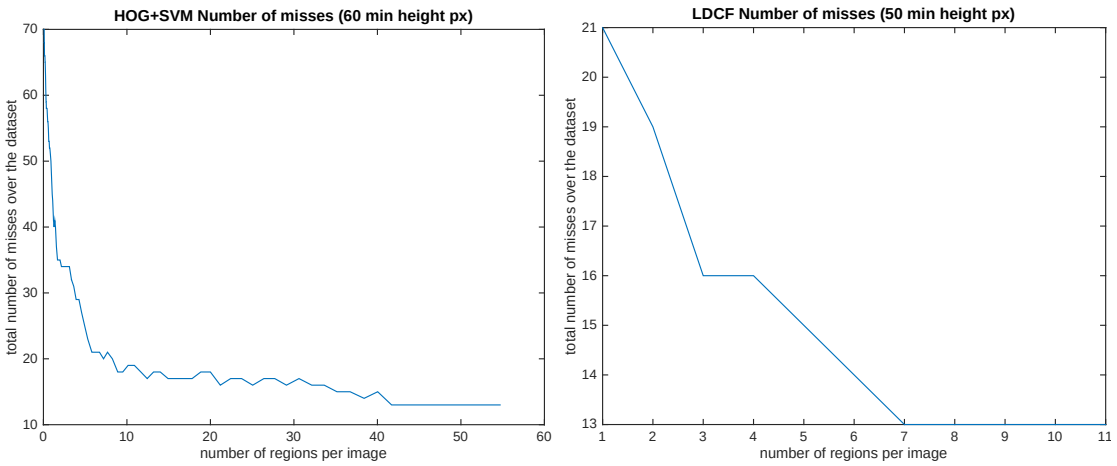


FIGURE 3.3: Total number of misses for both HOG and LDCF selector with average numbers of regions per image

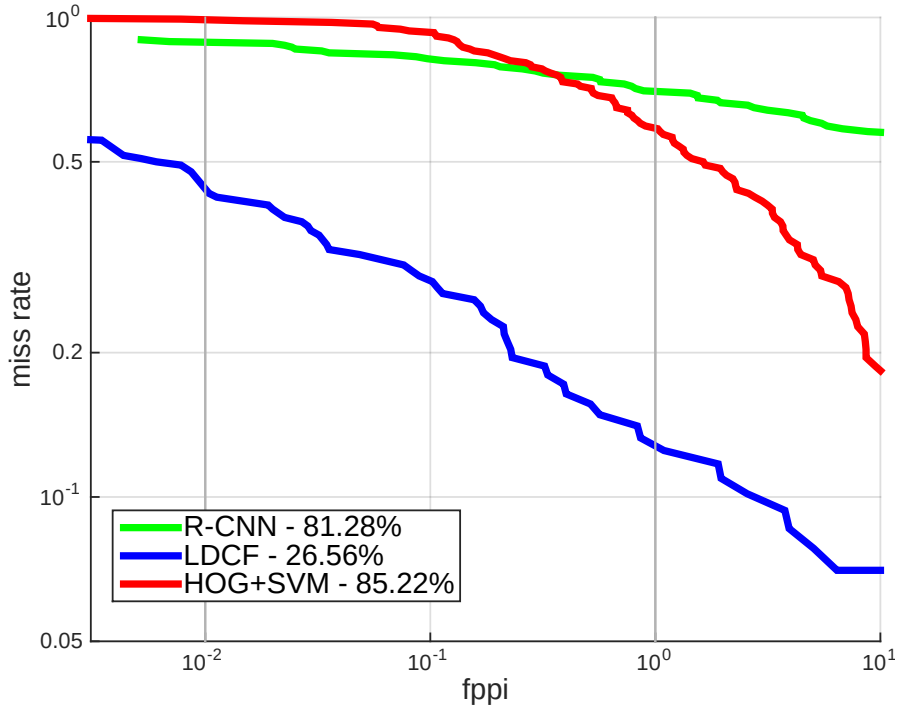


FIGURE 3.4: R-CNN ROC comparison with HOG and LDCF.
Value after the method's name represents the LAMR. Computed on set 06 of the test-set

3.3 Analysis of performance

Looking at the result and having analyzed how R-CNN is defined, it is possible to state that the selective search module is not the most suited method for the used dataset (Caltech Pedestrian Dataset Sec. 2.2). This because it returns class-independent region

proposals and performs well on images where there are few objects of interest placed in foreground.

With all the observations that we have done, the substitute of Selective Search for selecting region proposals should be a method that is able to produce a small number of proposals and that is able to suggest regions of interest reducing as much as possible the number of misses.

Chapter 4

Sliding Window CNN

According to the considerations done in Sec. 3.2, the first step is trying to see how CNN model trained by Ross *at al.* [30] behaves giving regions proposed by a sliding window approach: improvement in the results is expected.

4.1 Sliding Window

Sliding Window is the technique of moving a window along an image. This is done in order to be able to analyze subparts of the image, to extract some information.

Figure 4.1 shows the way this is done, starting from an initial window size, moving to a larger one.

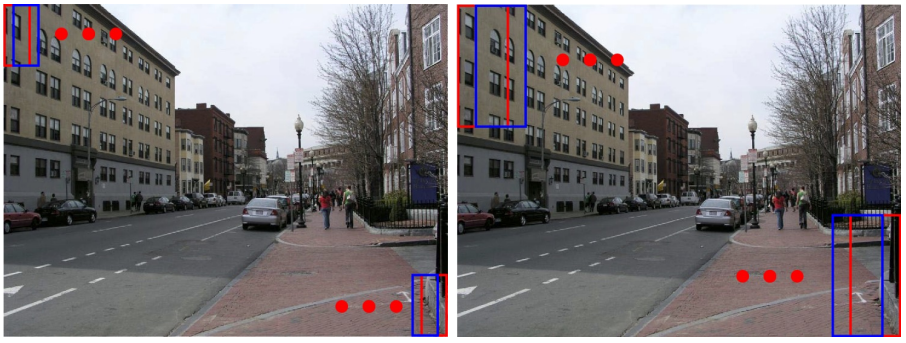


FIGURE 4.1: Sliding Window approach. Two different scales are shown.

The sliding window approach used as selector in this test has as parameters those defined in Table 4.1. In particular:

- **min size**: smallest window size used to generate proposals
- **max size**: largest window size used to generate proposals
- **scale step**: how much the window size is increased once the whole image has been analyzed at one scale
- **stride**: distance along the two dimensions by which the window is moved (Figure 4.1 shows in blue the red window shifted by a stride value on one dimension).

Region			
max size	min size	scale step	stride
200×100	50×25	1.1	10

TABLE 4.1: *Sliding window parameters*

When talking about sliding window, one very important parameter is the window **aspect ratio**. It is defined as $AR = \frac{w}{h}$, where w stands for width and h for height. After an analysis on the train-set (see Figure 4.2), this value has been set to $AR = 0.41$.

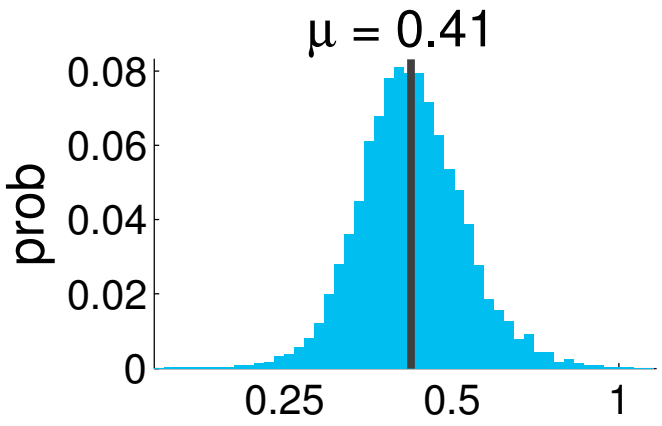


FIGURE 4.2: Distribution of bounding boxes aspect ratio [7]

4.2 Results

The analysis that has been done in Sec. 3 is confirmed by the results obtained using the sliding window approach (see Figure 4.3): using a selector that proposes almost all the regions containing pedestrians, there is a performance improvement. This improvement is extremely relevant, because by simply changing the selector, without any further training or finetuning, we have been able to move the log average miss rate **from** 81.28% **to** 61.99%. This is still far from the desired result, but shows how important the selector is.

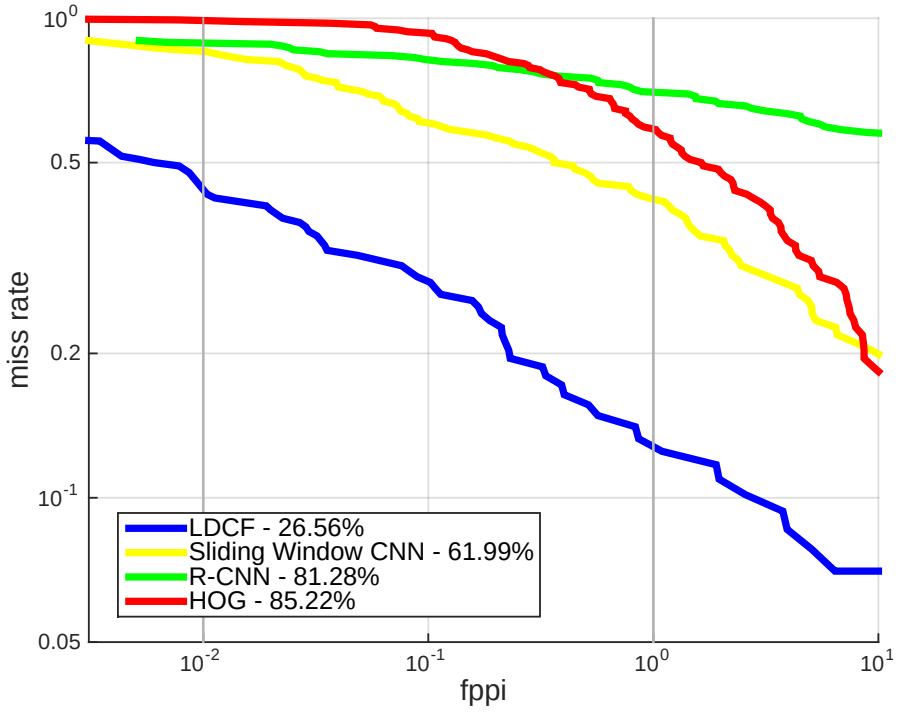


FIGURE 4.3: Sliding Window ROC. Overall comparison on set 06 of the test-set

4.3 Training SVM

A further improvement has then been obtained by training the SVM classifier, which takes as input the features proposed by the CNN model and returns as output the score associated to that regions. The higher such score, the more likely that region contains a pedestrian.

For training the SVM we have used the parameters defined in [33], using the Matlab toolbox for machine learning. These parameters are

- **Kernel funciton**: linear
- **Box constraint**: 10^{-3}

Results are shown in Figure 4.4.

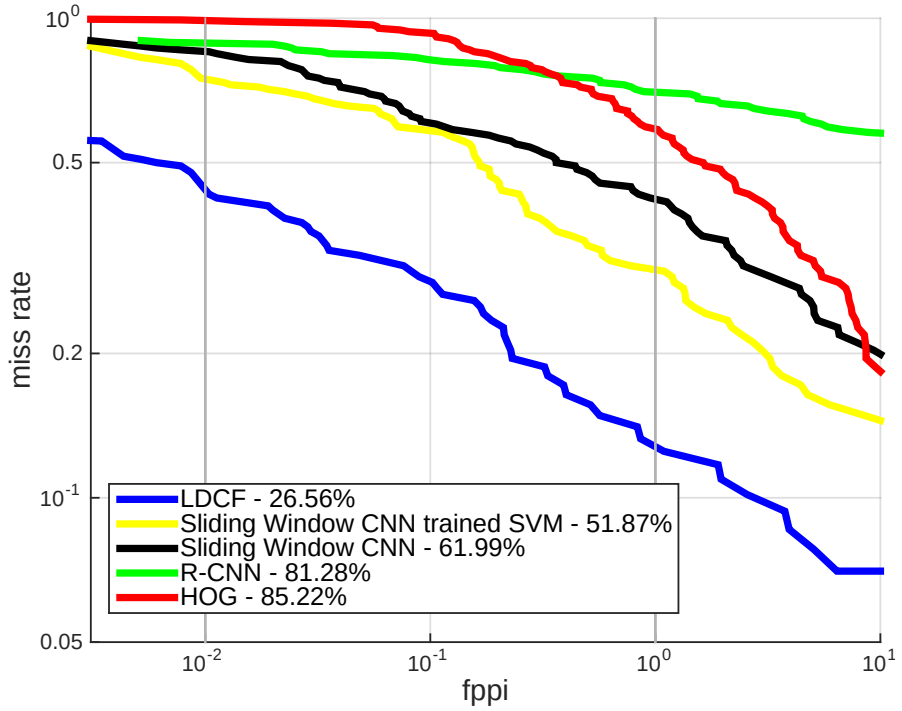


FIGURE 4.4: Sliding Window with trained SVM ROC. Overall comparison on set 06 of the test-set

Chapter 5

Ldcf-CNN

5.1 LDCF selector

In Sec. 3 we made an observation: Selective Search is not suitable as selector. This has then been confirmed by Sec. 4, showing how changing the selector led to a result improvement.

The draw-back of using a Sliding Window approach is that it generates too many window proposals and many of these are not regions of interest. Consequently, the next test involves the usage of a state-of-the-art method as a selector, which filters out the non-relevant data, producing fewer regions. The method that has been chosen is LDCF (see Sec. 1.4) since it is one of the best performing methods in pedestrian detection. Being LDCF our new selector, it satisfies all the conditions a search algorithm should be subject to, described in Sec. 1.6.1. Overview of Ldcf-CNN method is show in Figure 5.1.

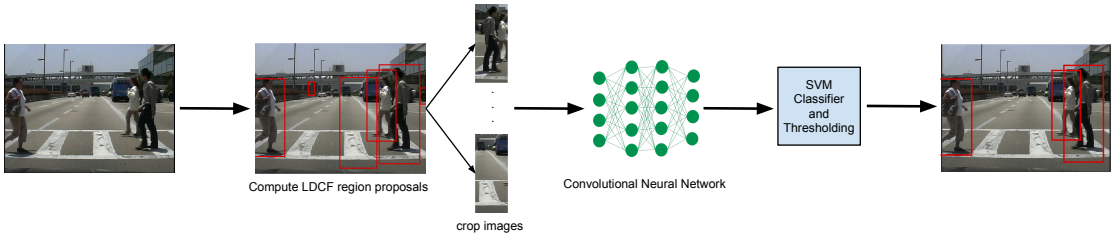


FIGURE 5.1: Overview LDCF-CNN

LDCF, given an input image, returns as output the bounding boxes and the associated scores. The scores are not considered, and all the LDCF region proposals are fed to the CNN. Its output features are finally given as input to the SVM, which computes for each *bounding box* the corresponding score.

5.2 Initial Results

By taking the R-CNN model trained on Pascal VOC dataset (see Sec. 2.1), training the SVM classifier on the features returned by the CNN (computed using the pipeline defined in Figure 5.1), the results are shown in Figure 5.2. As expected, there is an improvement with respect to the Sliding Window CNN approach (see Sec. 4.3).

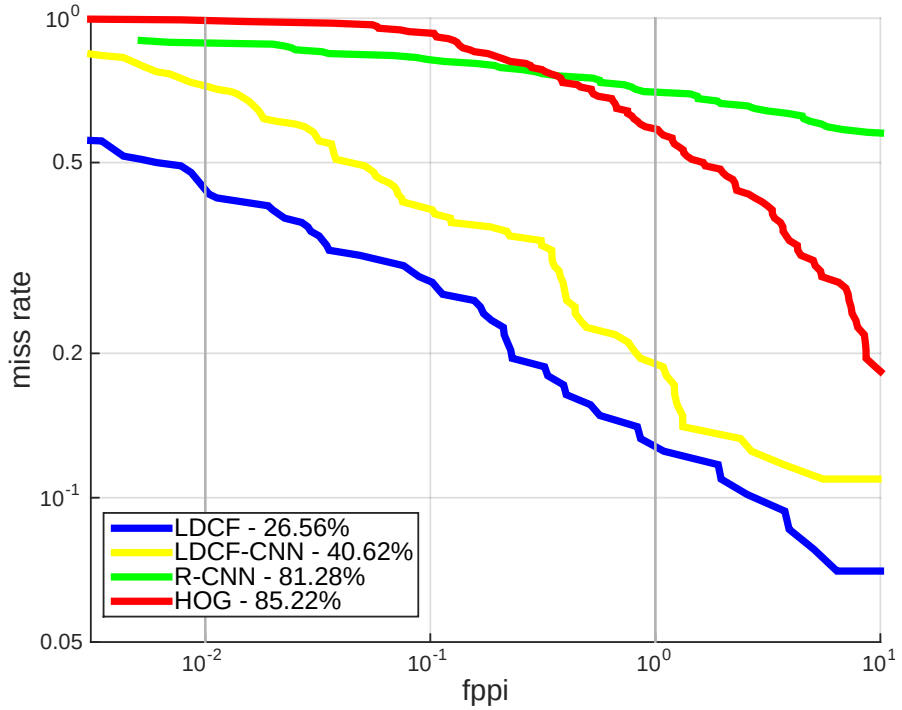


FIGURE 5.2: LDCF-CNN ROC. Overall comparison on set 06 of the test-set

The reason about this result is the number of region proposals: using the sliding window approach, all the possible regions of an image are input to the CNN, which computes the features, finally used by the SVM classifier to set the score. Without performing any kind of filtering, background regions with features similar to the pedestrian ones are given as input to the CNN, whose resulting features are hardly distinguishable by the

SVM, giving wrong scores: e.g. part of a Bus, under particular light conditions, could have features similar to a small pedestrian. The same could happen in the opposite case. Conversely, having a filter like LDCF before CNN, it filters out a relevant number of these particular regions, making the whole model more robust.

The region proposals with associated scores given by LDCF and LDCF-CNN are compared in Figure 5.3. As one would expect, matched region proposals (regions whose IoU with a ground truth pedestrian is larger than 0.5) should have large scores, while unmatched ones should have small scores. This is exactly what happen in both detectors, proving the correctness of our algorithm.

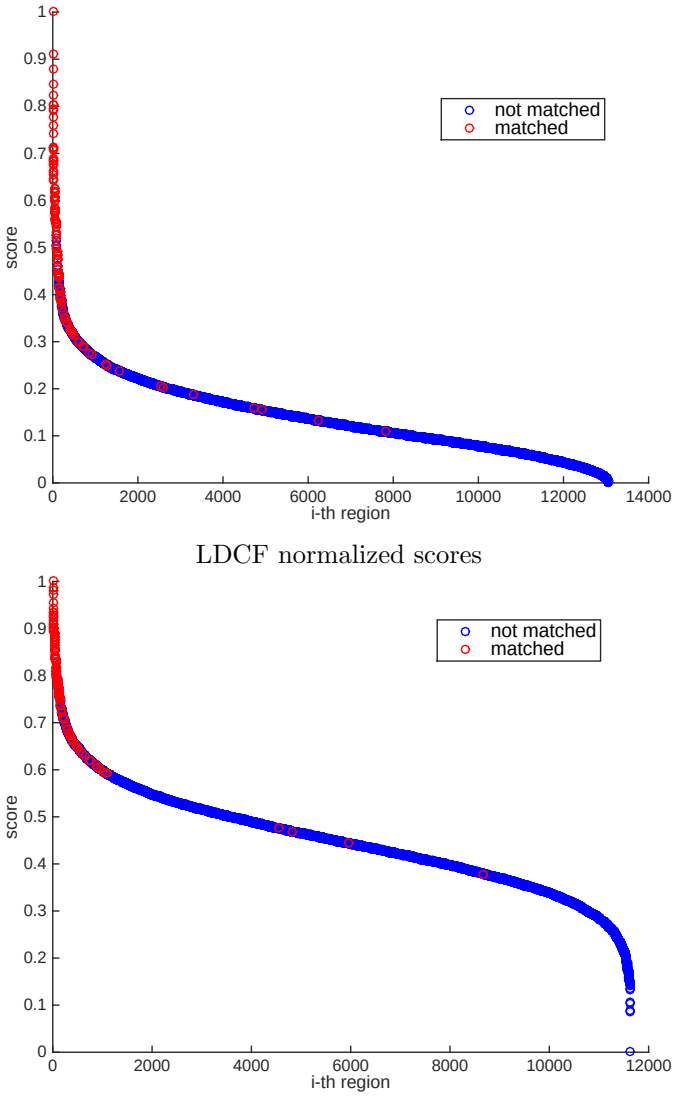
As already said, all the region proposals given by LDCF are input to the CNN, therefore one would expect to have the same number of regions in both detectors. This is not the case, and the mismatch is generated by a post-filtering algorithm called **Non-Maximum-Suppression**, applied only to the LDCF-CNN method. This algorithm allows to remove bounding boxes representing the same object. The idea is very simple: if two bounding boxes have an Intersection-over-Union value that is greater than a certain threshold, we consider them representing the same object, and only the one with the highest score is kept. This lead to more robust and accurate results.

5.3 Finetuning

Fine-tuning refers to circumstances when the parameters of a CNN model must be adjusted very precisely in order to agree with the observations. We start from a model, and from there we change the value of the model to best fit the data in our dataset.

5.3.1 Caffe

Caffe is a deep learning framework developed by Berkeley Vision and Learning Center and by community contributors. It provides a clean and modifiable framework for state-of-the-art deep learning algorithms and a collection of reference models.



LDCF normalized scores

LDCF-CNN normalized scores

FIGURE 5.3: LDCF and LDCF-CNN normalized score analysis to prove code correctness

By separating model representation from actual implementation, Caffe allows experimentation and seamless switching among platforms for ease of development and deployment from prototyping machines to cloud environments.

The model representation is defined using a particular format, called prototxt. In this file is possible to define a variety of properties

- size and output of each layer

- type of layer: Convolution, Relu, Pooling, Dropout, Inner product
- kernel size
- weight decay parameter
- learning rate parameter
- how to initialize the weights of each layer
- which are the data to use for training
- many others more

These parameters allow not only to define the network structure, but also give a deep control over all the training aspects.

Caffe requires also preprocessed data in the LMDB format. LMDB (Lightning Memory-Mapped Database) is an ultra-fast, ultra-compact key-value embedded data store that uses memory-mapped files, so it has the read performance of a pure in-memory database while still offering the persistence of a standard disk-based database.

5.3.2 Parameters

For the finetuning process, the R-CNN model (see Sec. 1.6) has been used as a starting model, from which the weights are taken. In Figure 5.4 the AlexNet structure is represented, showing all the layers' size and type.

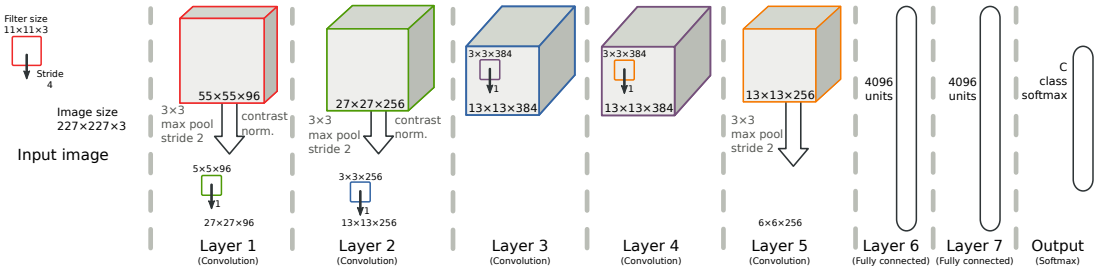


FIGURE 5.4: AlexNet architecture representation

The activation function used by this model is the **Rectified Linear Unit (ReLU)** one. This is an approximation of the analytic function $f(x) = \ln(1 + e^x)$ called **softplus** function (see Figure 5.5) and it's defined as

$$f(x) = \begin{cases} x & \text{if} \quad x > 0 \\ 0.001x & \text{otherwise} \end{cases} \tag{5.1}$$

The use of the ReLU [34] activation function, has been shown to enable training deep supervised neural networks without requiring unsupervised pre-training. Rectified linear units, compared to sigmoid function or similar activation functions, allow for faster and effective training of deep neural architectures on large and complex datasets, and this the reason why they are used.

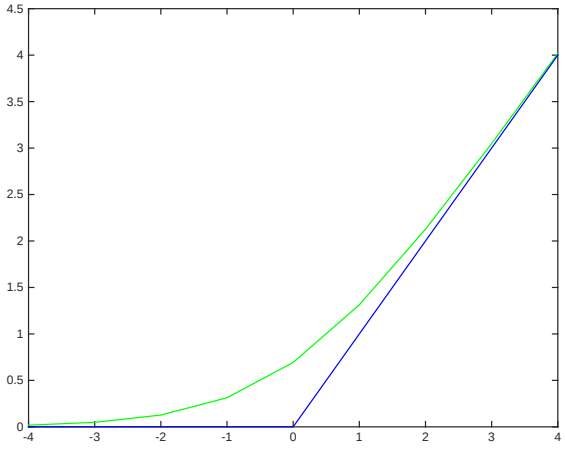


FIGURE 5.5: Plot of the ReLU (blue) and Softplus (green) functions near $x = 0$

About the other parameters, like learning rate and weight decay, these are described in Table 5.1. As it's possible to see, a learning rate multiplier and a weight decay multiplier are defined for each layer. To obtain the final coefficient values, the product between the layer value and the global value must be performed: multiplier$*$global value, e.g. (learning rate multiplier layer 1) $*$ (base learning rate). This configuration allows to change the global values after a fixed number of iterations, affecting the learning mechanism over all the layers.

One important aspect to be aware of, is that in order to perform finetuning, some structure changes must be applied to the R-CNN model. In particular, knowing how

		conv1	conv2	conv3	conv4	conv5	inner1	inner2
learning rate mult	weights	1	1	1	1	1	1	1
	bias	2	2	2	2	2	2	2
weight decay mult	weights	1	1	1	1	1	1	1
	bias	0	0	0	0	0	0	0
weight decay		0.0005						
base learning rate		0.001						

TABLE 5.1: *R-CNN layers' parameters*

the weights are updated (see Back-propagation described in Sec. 1.5), three additional layers are required. These are used just during the finetuning phase and will be discarded for the testing part, which are

- **InnerProduct** layer with a number of outputs equivalent to the number of classes: in this case just two, pedestrian - not pedestrian
- **Accuracy** layer used for computing the classification accuracy for a one-of-many classification task. This is used to show how the model performs over the finetuning iterations.
- **SoftmaxWithLoss** layer drives learning by comparing an output to a target and assigning cost to minimize. It computes the multinomial logistic loss for a one-of-many classification task, passing real-value predictions through a softmax to get a probability distribution over classes.

The new InnerProduct layer has to be initialized: the weights are initialized using a Gaussian distribution with standard deviation 0.01, while bias is initialized with 0.

Hereinafter is shown the prototxt code for adding the last layers to the R-CNN structure.

```
layer {
  name: "fc8_Caltech"
  type: "InnerProduct"
  bottom: "fc7"
  top: "fc8_Caltech"
  param {
    lr_mult: 1
    decay_mult: 1
```

```
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  inner_product_param {
    num_output: 2
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "fc8_Caltech"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "fc8_Caltech"
  bottom: "label"
  top: "loss"
}
```

5.3.3 Model identification

To be able to perform finetuning, data must be preprocessed as described in Sec. 5.3.1, defining what are training and validation sets. According to [33], the training set is split as in scenario A (see Sec. 2.2.1) for the so called **hold-out** cross validation:

- **set00** − **set04** train
- **set05** validation

During the finetuning process, a log.txt file is created containing information such as **accuracy** and **loss** values, computed using the model trained at the i-th iteration over the validation set. Here a distinction has to be pointed out

- **iterations**: Number of batches that have been analyzed. E.g. if we have three batches and 6 iterations have been performed, then each batch has been analyzed twice. If the number of iterations is not a multiple of the number of batches, not all the images will be analyzed.
- **epochs**: it's defined as number of iterations/number of batches. E.g. if we have 2 epochs, this means that independently of the number of batches, each of them will be analyzied twice.

About the loss value: consider a model f mapping inputs x to predictions $y = f(x) \in Y$. Let t be the **true label** of input pattern x. Then a loss function $L : Y \times Y \to \mathbb{R}$ measures the quality of prediction.
Now call a collection of observations $S = \{(x_1, t_1), \ldots, (x_N, t_N)\} \in (X \times Y)^N$ and corresponding predictions $y_1, \ldots, y_N$, then the loss function is the overall error, defined as

$$L = \sum_n l(y_n, t_n) \quad \text{where} \quad l(x_n, t_n) = -log(P(Y = t_n | x_n, W, b)) \tag{5.2}$$

Therefore, the loss value can be used to understand where to stop with the finetuning iterations. An example loss function, is shown in Figure 5.6. In this case the good iteration would be the 1000th one.

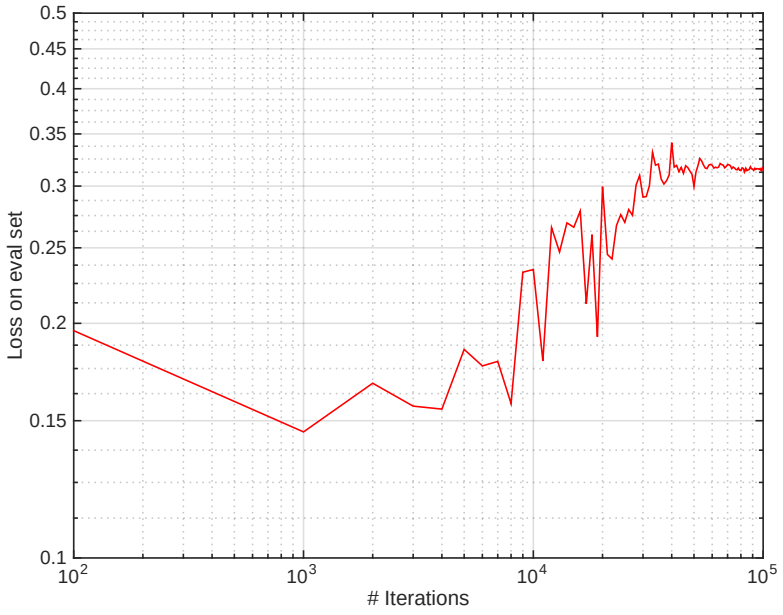


FIGURE 5.6: Example of loss function. Model at iteration 1000 is the one that produces the lowest loss value over the validation set.

5.4 Data manipulation

The finetuning process is extremely sensible to the data, therefore is essential to find which are the transformations to be applied and which types of data to select, in order to find better results.

5.4.1 Negative-Positive ratio

The number of negative region proposals is selected such that there are

- 5 negative LDCF region proposals, each
- ground truth pedestrian

This has been proved by Hosang *et al.* [33] to be a ratio that produces good results. By doing this, the CNN has enough information to learn pedestrian's features, and is also able to better learn LDCF region proposals that represent something different from pedestrians.

Using the same approach of analysing the loss function through all the iterations to identify the best model, the results are shown in Figure 5.7.

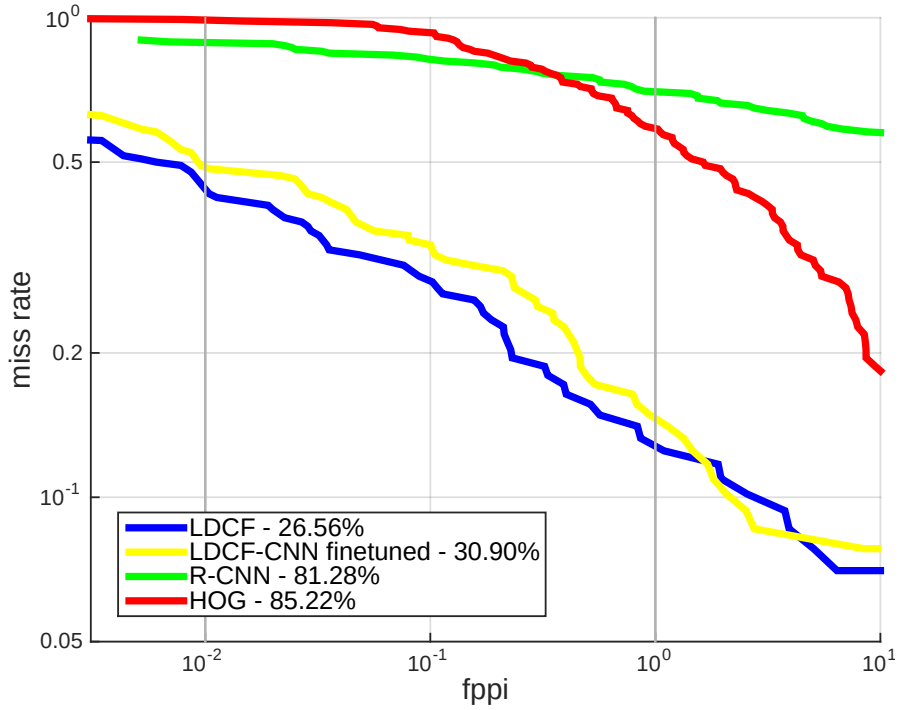
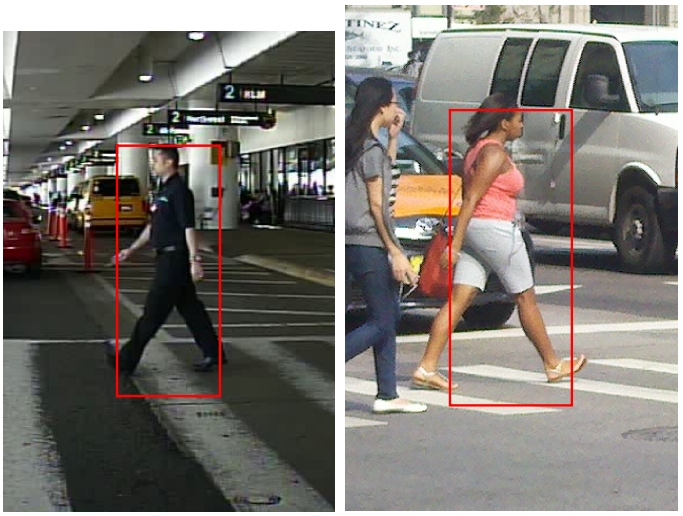


FIGURE 5.7: LDCF-CNN ROC. Overall comparison after finetuning the model with Ldcf negative region proposals and ground truth regions. Computed on set 06 of the test-set

5.4.2 Padding

An important observation can be directly derived by looking at Figure 5.8. Here it is possible to see one correctly identified bounding box, and another slightly shifted one. What is needed is a mechanism that could be used to simulate during the finetuning phase the selector "uncertainty". The solution is found in padding the regions, followed by random cropping mechanism.

To be able to correctly pad the regions, an analysis over the dataset is required. Measuring the amount of shift over the matched LDCF region proposals, comes out that the average padding is about 17 pixels (see Figure 5.9).



Correct BB Shifted BB

FIGURE 5.8: Uncertainty in BB proposals by the selector

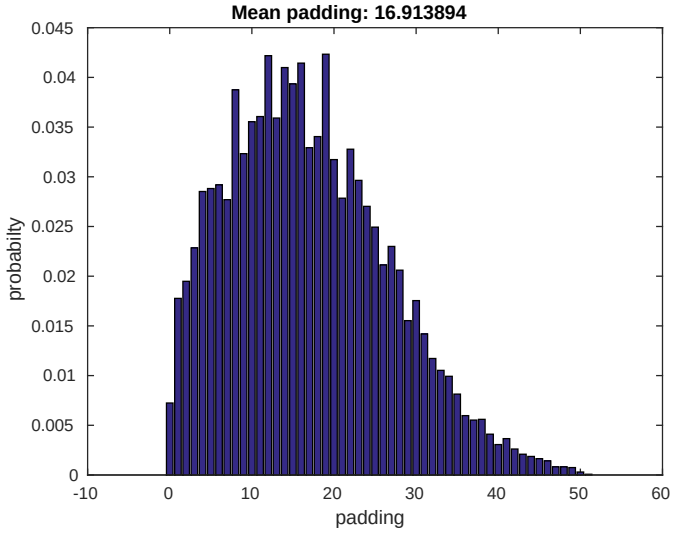


FIGURE 5.9: Caltech Dataset padding distribution using LDCF generated region proposals over the train set.

At this point, what it's done for padding the region is to

- take every region proposal
- add x pixels of padding such that when the region is warped to a 256×256 size image, the final padding is 16 pixels
- crop from the 256×256 size region a 227×227 size region used for finetuning (see Figure 5.10)

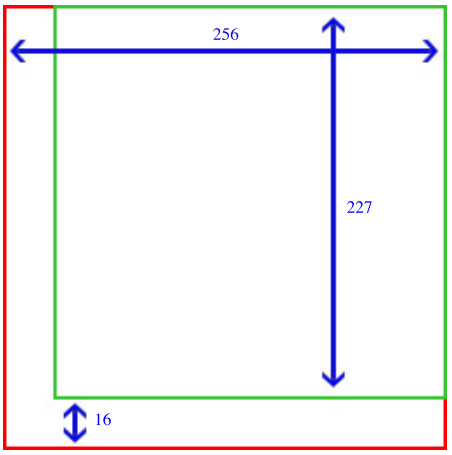


FIGURE 5.10: Random cropping of 227×227 size from the given image

In this way, padding is added to the region in a suitable way. E.g. suppose to have a region of size 20×50 pixels, the added padding is proportional the the region size without disrupting the region content.

Padding followed by random cropping (see Figure 5.11) has the effect of reducing overfitting: fitting the training data, reducing ability of predicting unseen data. the region proposals are now less "precise" in selecting the pedestrian, simulating what is the selector behaviour with unobserved images.

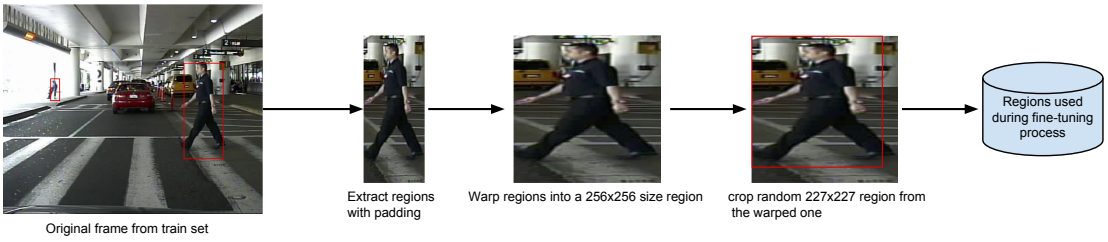


FIGURE 5.11: Generate images for finetuning process

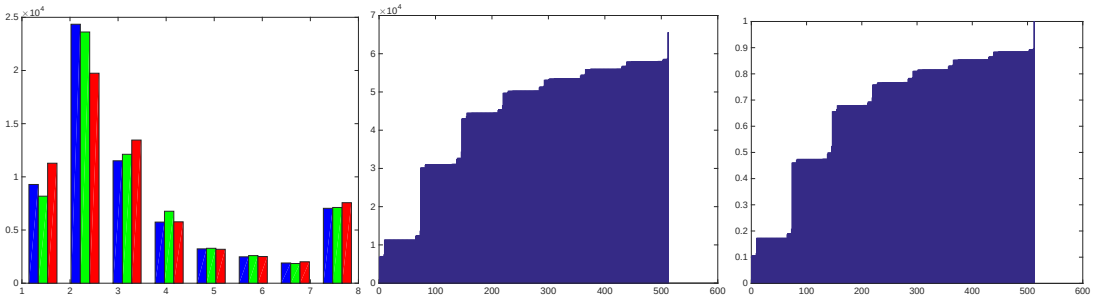
5.4.3 Data decorrelation

To create the final dataset used during the fine-tuning process, positive and negative regions have to be selected. According to the criteria define in Sec 5.4.1, every positive regions, 5 negative regions are selected among all the proposed ones. There are a variety of methods that one could use. Here is proposed one based on **histogram**, that tries decorrelate data at most.

For each image in the dataset, the histogram is computed using 8 bins for each of the RGB channels, resulting in a final $H \in \mathbb{R}^{8\times8\times8}$ matrix. After that, the matrix is vectorized and finally normalized (see Figure 5.12). This is done to make the feature vector more robust: e.g. more robust to light intensity variations.

In the H matrix, the value v of the element in position i, j, k represents v pixels in the image, having

- R value in the range $\Big[(i-1)*32 \quad , \quad i*32\Big]$ for $i \in [1, 8]$
- G value in the range $\Big[(j-1)*32 \quad , \quad j*32\Big]$ for $j \in [1, 8]$
- B value in the range $\Big[(k-1)*32 \quad , \quad k*32\Big]$ for $k \in [1, 8]$



(A) RGB histogram (B) Vectorized matrix (C) Normalized vector

FIGURE 5.12: (A) RGB histogram of a region proposals which is used to compute the H matrix; (B) H matrix is vectorized; (C) The normalized vector represents the features for that region proposal;

Once the vectorized and normalized version of the H matrix is computed, this vector is considered as the set of features representing the region proposals.

Call $F \in \mathbb{R}^{N\times512}$ the matrix for the dataset, where

- N: the number of region proposals extracted from the images in the dataset
- 256: size of the features vector for each region (given by $8 \times 8 \times 8 = 512$)

a greedy algorithm based on the **Feedforward Subject Selection** is used to identify which among the regions has to be used.

Given the F matrix, compute the D matrix, where the i, j element represents the Euclidean distance between the features of region i and j. This matrix will be later used by the algorithm. The properties of this matrix are

- Square
- Symmetric
- Zeros on the diagonal

The main steps of the greedy algorithm are as follows

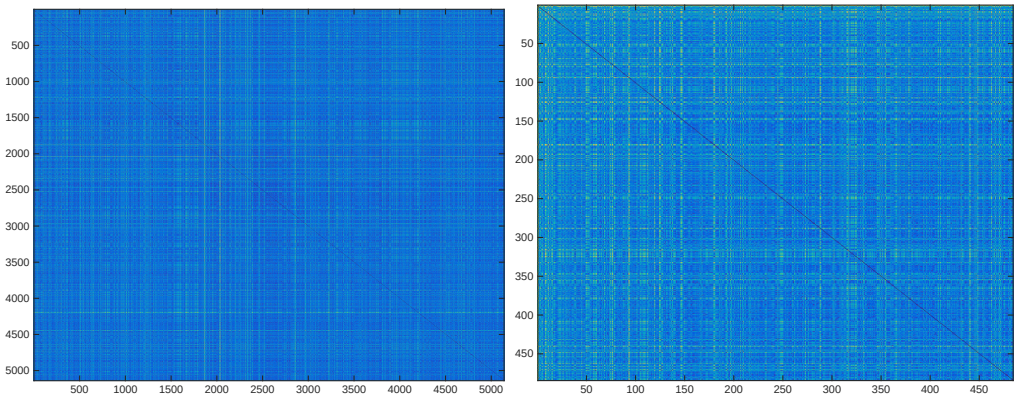
1. Take i-th row/column of input matrix whose sum of elements is the largest one
2. Remove from the input matrix the i-th row and i-th column
3. Repeat until the desired number of regions has been selected from the dataset.

Hereinafter the code of the algorithm

```
function idx_selected_regions = greedy_selection( corr_mat, num_regions )

  total_corr = sum(corr_mat,2);
  [~, idx] = min(total_corr);
  idx_selected_regions = [idx];

  for i = 2 : num_regions
      sub_corr_mat = corr_mat(:, idx_selected_regions);
      % set to +inf in order to keep the correct
      % indices for the following selections
      sub_corr_mat(idx_selected_regions,:) = inf;
      sub_total_corr = max(sub_corr_mat,[],2);
      [~, idx] = min(sub_total_corr);
      idx_selected_regions = [idx_selected_regions; idx];
  end

end
```

Calling the **greedy_selection** function with corr_mat $= -D$, the result is a set of indices of the mostly decorrelated regions. By considering a small subset of images belonging to the dataset (for a computational reason), the resulting decorrelation after applying the greedy algorithm is visible in Figure 5.13. This is the plot of the distance matrix where

- dark means correlated (large value)
- bright mean decorrelated (small value)



(A) Distance matrix for all regions in the subset

(B) Distance matrix of selected regions in the subset

FIGURE 5.13: (A) Distance matrix computed over all the regions of the first subset of the train set: average L^2 distance between feature vectors **3.67**; (B) Distance matrix computed by taking from A only the selected regions: average L^2 distance between feature vectors **4.71**.

As expected, all the elements on the diagonal are zero and the plot shows the effect of the defined algorithm.

Due to the computational effort required by this method, a different and simpler approach is used: random selection of the desired number of regions. Even if this algorithm has not been used, it still remains a very good alternative, that could lead to better results.

5.5 Results

By using all the previously defined methods (padding, cropping and and random selection) the finetuning process has produced the results shown in Figure 5.14.

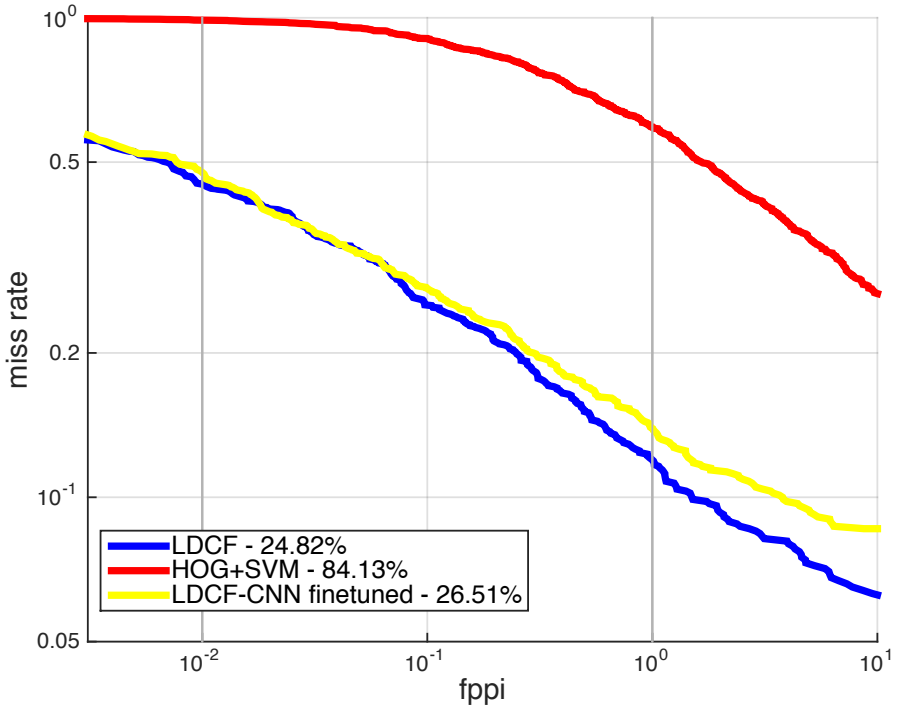


FIGURE 5.14: Ldcf-CNN best result: ROC overal comparison

The obtained results prove that Ldcf-CNN performs similar to state-of-the-art methods like Ldcf.

5.5.1 Softmax vs. SVM classifier

All the results have always been computed using a SVM linear classifier, taking the features returned by CNN as input, producing the scores

$$scores = W * features + b \tag{5.3}$$

This is not really necessary, because the network already provides a Softmax layer for classification.

The SVM classifier is more *local objective* than Softmax, because it considers only the support vectors, which are used to compute the final model. Conversely, Softmax classifier considers always all the data. In other words, SVM would be indifferent to the score of samples that satisfy the margin, while Softmax would like the correct classes to have

always a higher probability and the incorrect classes always a lower probability. The results of using both SVM and Softmax as a classifier is shown in Figure 5.15, proving that SVM performs slightly better.

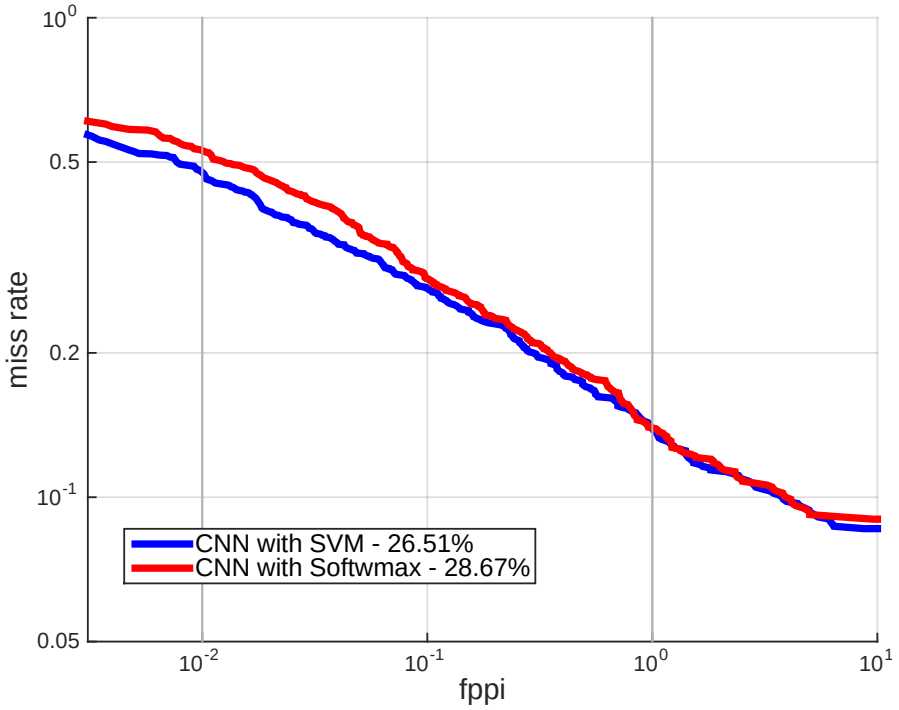


FIGURE 5.15: ROC CNN model with Softmax and SVM classifiers

5.6 K-Folds cross validation

The obtained result starts getting really close to LDCF, but still it doesn't perform well enough.

During all the tests, the set 05 has always been used as the validation set in the hold-out cross validation, being the one used to identify which among all the models computed during finetuning process (one per each iteration) had to be selected. However, having additional information usable during the training phase, would lead to a more accurate model, estimated on more data. The solution of this problem is found is the **K-folds** cross-validation.

In k-fold cross-validation, the original dataset is randomly partitioned into k equal sized subsamples. Of the k subsamples, a single subsample is retained as the *validation set*

for testing the model, and the remaining $k-1$ subsamples are used as *training data*. The cross-validation process is then repeated k times (the folds), with each of the k subsamples used exactly once as *validation set*.

In this case, having the train set composed by 6 sets, a 6-folds cross validation is performed. The 6 loss-functions computed over the train-set are shown in Figure 5.16.

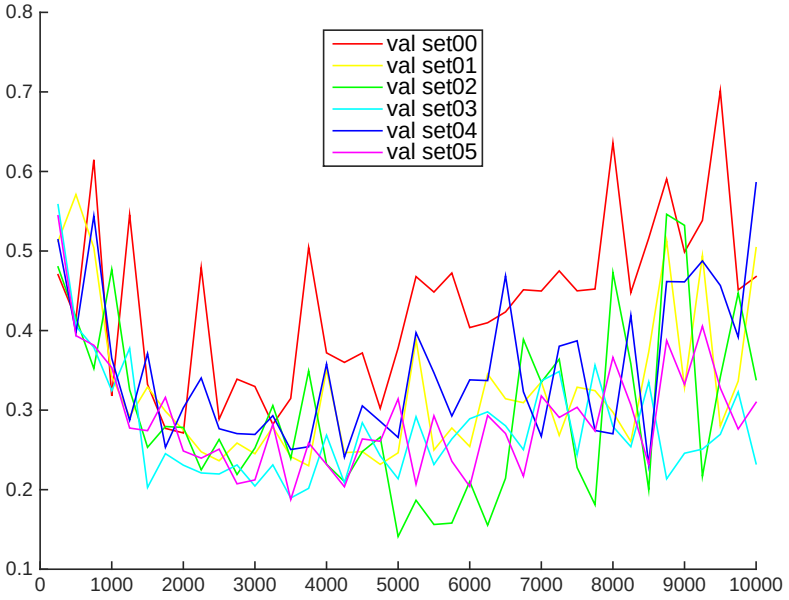


FIGURE 5.16: K-folds loss function for each fold

As it is possible to see, the behaviour consistently changes using one validation set rather than the others. The k results from the folds are then averaged to produce a single estimation used to find the best model, which can be seen by looking at Figure 5.17. Using this function, the finetuned process is then repeated for the last time, using instead the whole train set.

The average loss function is just an estimation, rather than the exact behaviour, of how our model should perform with unseen data, therefore it may not happen that a minimum in the loss function corresponds to the best result over the test set. Furthermore, the function has been derived by computing the performance over a subset of all the identified models (for practical reasons).

Having said that, by looking from iteration 2500 to 3000 it is possible to see how the model performance is almost stationary, and by smoothing the function (removing picks

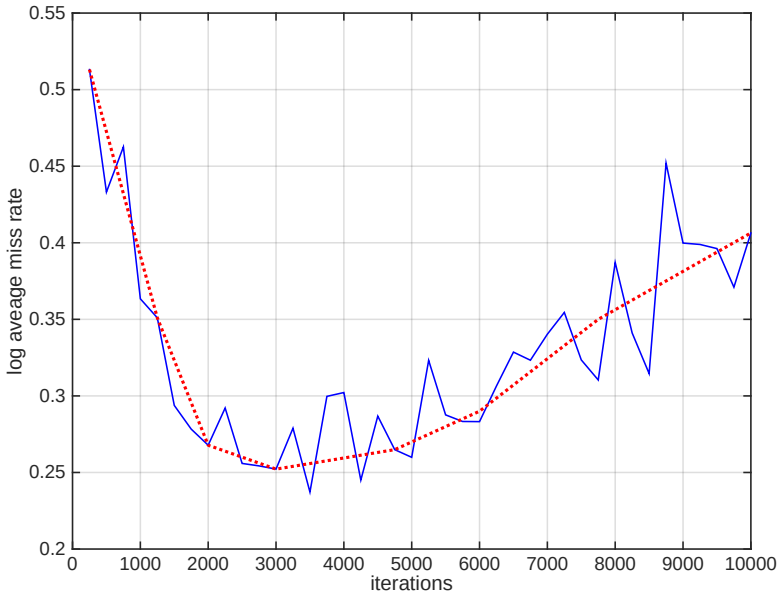


FIGURE 5.17: K-folds average loss function. In red the smoothed version.

due to noise), this stationarity would proceed till iteration 5000, where the loss value then starts increasing again. This situation means that while we are getting almost the same results, we are loosing in generality over unseen data, because proceeding with the iterations we get more and more specialized in classifying the training data (**overfitting**). This results in getting worse results on the test set.

Among the models, especially those in the [2500-5000] iterations range, the one at iteration **3000** is chosen (with $loss = 0.2522$). The result of using this model with our method, is visible in Figure 5.18.

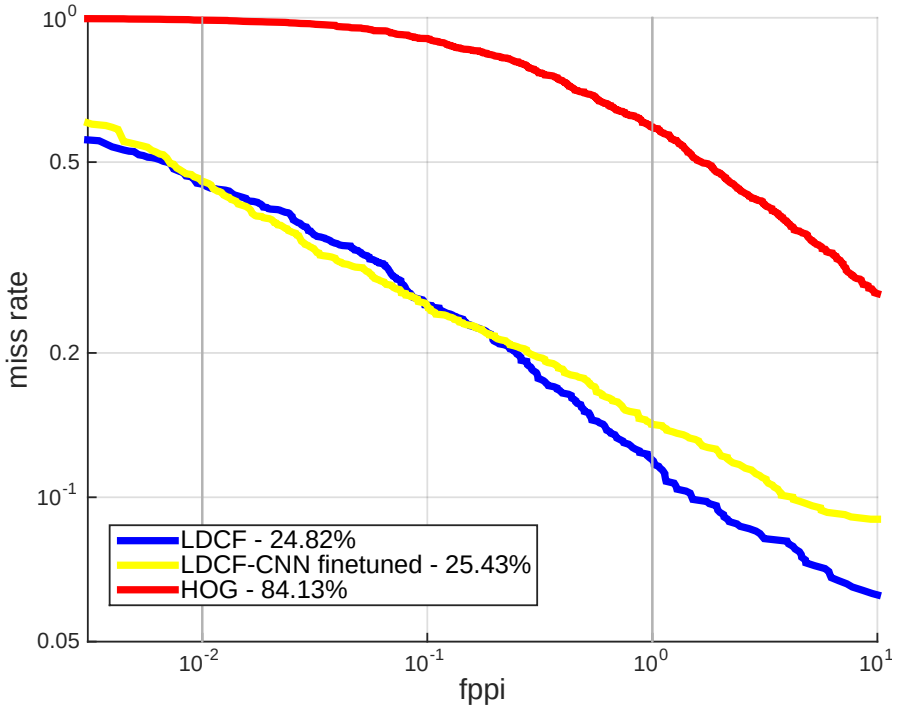


FIGURE 5.18: LDCF-CNN with model identified using k-folds

5.7 Thresholding

Throughout all the test that have been performed, one fact clearly emerged: CNN is not good enough in classifying pedestrians under particular conditions, like those that are far from the camera, in the shadow, etc. because the features extracted from the regions containing them are very similar to the ones extracted from some background regions. Even if there are few pedestrian satisfying the latter conditions, the opposite situation is different: there is a relevant number of background regions with misleading features that are interpreted in the wrong way by the classifier. To solve this problem, the information generated by selector could be used, without needing additional computations.

The selector (LDCF), computes all the bounding boxes proposals giving at each of them the related score (as described in previous sections 5.1). These scores are then discarded and all the identified proposals, whether they are good or bad, are given as input to the CNN. Given this situation, one very simple solution is the following one: use the selector's score to filter out all the proposals that the selector itself identifies as "very unlikely to be a pedestrian", sending only the remaining proposals to the CNN model.

At this point, the only parameter that has to be identified is the thresholding value to be used to filter out those regions.

In Figure 5.19 is represented the log average miss rate value on the *set-05*, achieved using the threshold value defined on the horizontal axis on LDCF before sending the regions to the CNN. The steps are:

- Select regions using the selector
- Filter-out all those regions whose score is less than the identified threshold
- Pass all the remaining regions to the CNN for scoring them.

Threshold 85 is the one that produces the lowest log average miss rate.

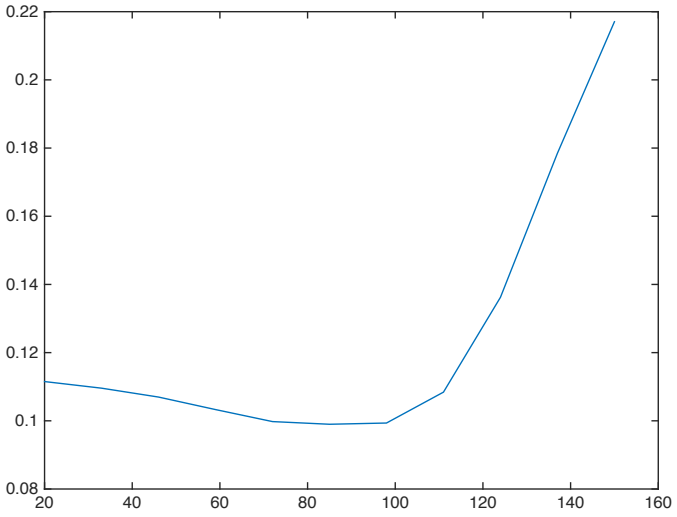


FIGURE 5.19: Thresholding: log average miss rate on set-05, achieved using a certain threshold value on the selector of LDCF-CNN method.

Filtering out the selector's proposals according to the threshold and sending the remaining proposals to the CNN model, lead at the result shown in Figure 5.20. **This proposed LDCF-CNN method performs better than the state-of-the-art LDCF method alone, gaining more than 2.3% in terms of log average miss rate.**

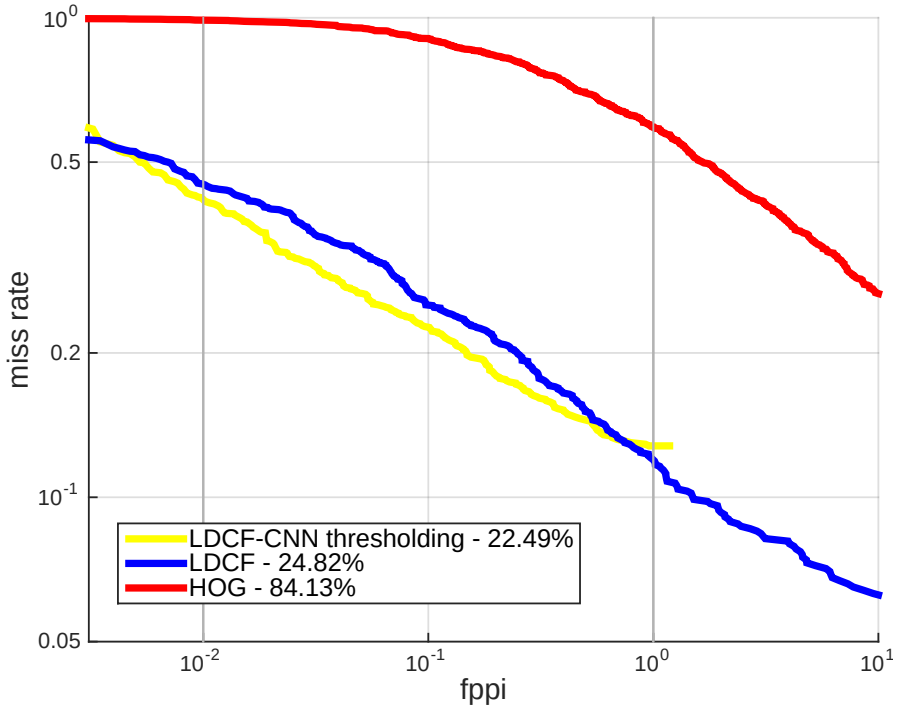


FIGURE 5.20: LDCF-CNN model identified using k-folds with thresholding on selector

5.8 Profiling

In Sec 1.4 the LDCF detector is briefly reviewed, but what is missing is to understand where the resources are allocated for potential features improvements. Table 5.2 [1] shows the two main functions that compose the LDCF method.

Function	**Exec time**	**Data**	
		IN	**OUT**
Sliding window object detection	357 ms	900 KB	20 MB
Sliding window classifier	6 ms	20 MB	0.21 KB

TABLE 5.2: *LDCF profiling*

As it's possible to see, the most demanding part is the one addressing the feature extraction. If one would speed-up the selector, that part needs to be carefully changed.

[1] These measurements have been taken on an intel core i-7 3 GHz processor with Matlab code, for a single image

With the same goal of LDCF profiling, the CNN architecture is analyzed (see Table 5.3 [2]) to understand which are the most demanding layers. The last column shows the storage demand for each layer.

		Operations		Time	Total time	Weights Bytes
layers	**conv1**	convolution relu pool norm	4.62 ms 0.452 ms 0.457 ms 0.848 ms	6.337 ms	6.337 ms	136.5 KB
	conv2	convolution relu pool norm	5.832 ms 0.271 ms 0.373 ms 1.489 ms	7.97 ms	14.344 ms	1.173 MB
	conv3	convolution relu	8.331 ms 0.288 ms	8.619 ms	22.964 ms	3.377 MB
	conv4	convolution relu	6.926 ms 0.26 ms	7.186 ms	30.150 ms	2.53 MB
	conv5	convolution relu pool	4.02 ms 0.241 ms 0.258 ms	4.52 ms	34.670 ms	1.689 MB
	fc6	inner prod. relu	45.923 ms 0.617 ms	46.540 ms	81.210 ms	144.01 MB
	fc7	inner prod. relu	13.778 ms 0.497 ms	14.274 ms	95.484 ms	64.02 MB

TABLE 5.3: *CNN profiling*

The graphical representation of the computation time for each layer, are shown in Figure 5.21 and 5.22. The two fully connected layers together define the final model size, therefore by testing them, trying to reduce their size or the weights format, it's possible to have a lighter model.

The overall model runs in GPU mode on a NVIDIA JETSON TK1 board, processing one image in 405.011 ms, therefore 2.47 fps.

[2]These measurements have been taken in GPU mode, on a NVIDIA JETSON TK1 board, for a single region of size 227×227

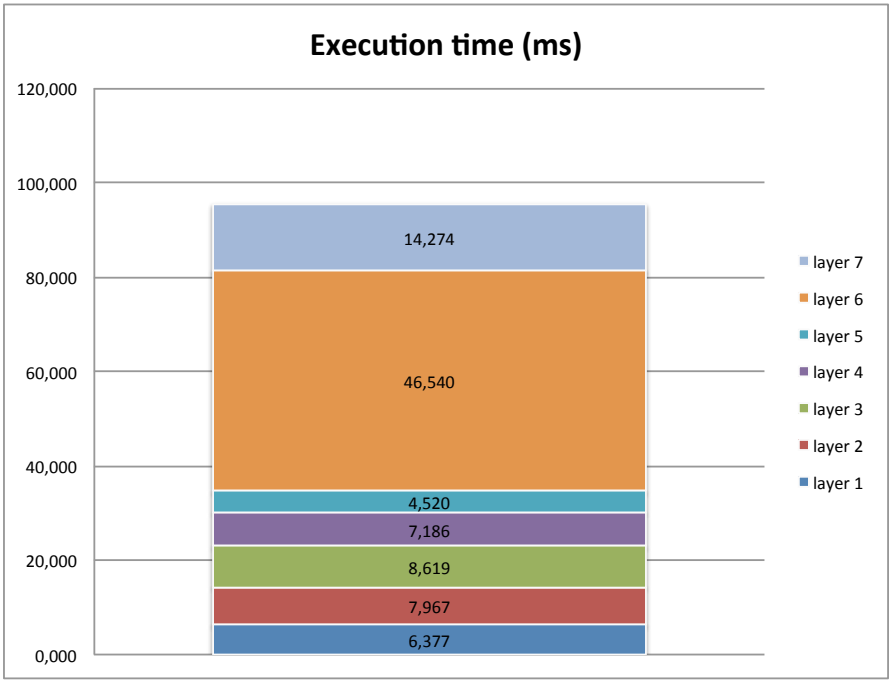


FIGURE 5.21: Execution time of each layer in ms in GPU mode on a NVIDIA JETSON TK1 board.

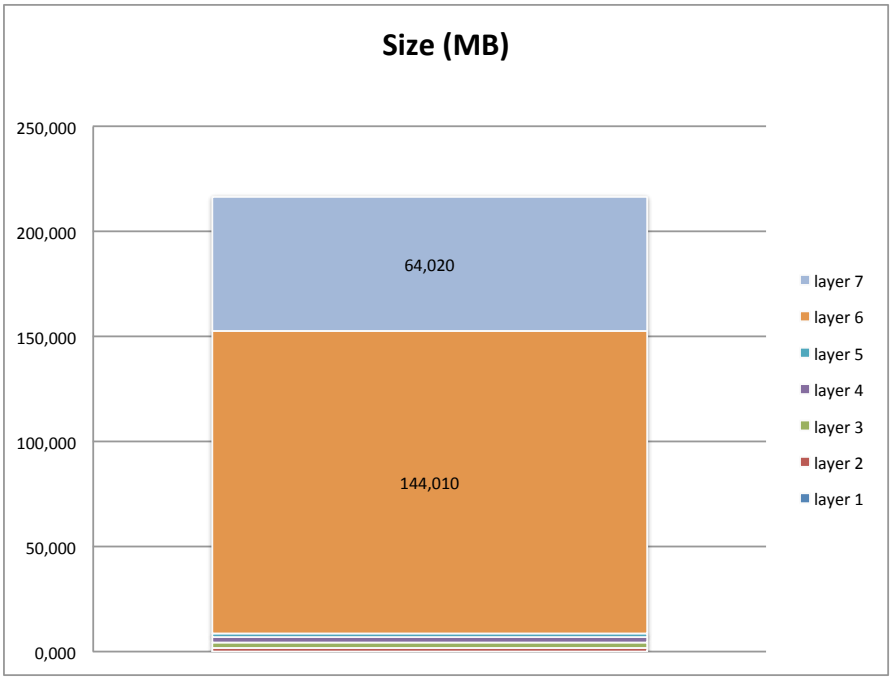


FIGURE 5.22: Size of each layer expressed in MB

Chapter 6

Model size reduction

ST Microelectronics has defined the goals of the second part of the thesis. In particular, they use ACF as their own detector, therefore all the tests will be done by taking the identified model, and use it to prove that despite the selector, CNN improves the performance in terms of accuracy (reducing the log average miss rate).

In Figure 6.1 the results of applying directly the model with ACF selector. Without applying any kind of thresholding (as described in Sec 5), CNN improves the results by **reducing the log average miss rate value of** 3.58%.

In order to be able to apply the **Analyze-Then-Compress paradigm** (ATC) where the node is responsible for analyzing the data, the model is required to be small, to fit the hardware constraints. For this reason, other architectures are also analyzed.

6.1 Network In Network

Network In Network [35] is a novel deep network structure, where micro neural networks are built between the convolutional layers, to abstract the data within the receptive field.

Convolutional neural networks consist of alternating convolutional layers and pooling layers. Convolution layers take inner product of the linear filter and the underlying

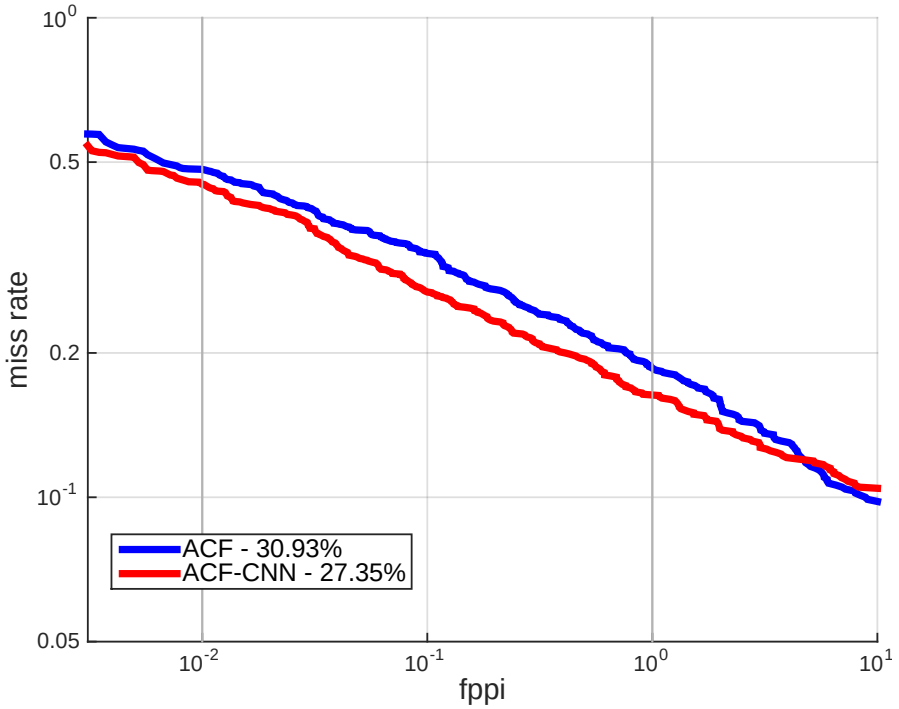


FIGURE 6.1: ROC: ACF vs. ACF-CNN

receptive field followed by a nonlinear activation function at every local portion of the input.

The convolution filter in CNN is a generalized linear model (GLM) for the underlying data patch, and by replacing the GLM with a more powerful nonlinear function approximator, can enhance the abstraction ability of the local model. In NIN, the GLM is replaced with a "micro network" structure called mlpconv layer, which is a general nonlinear function approximator (see Figure 6.2).

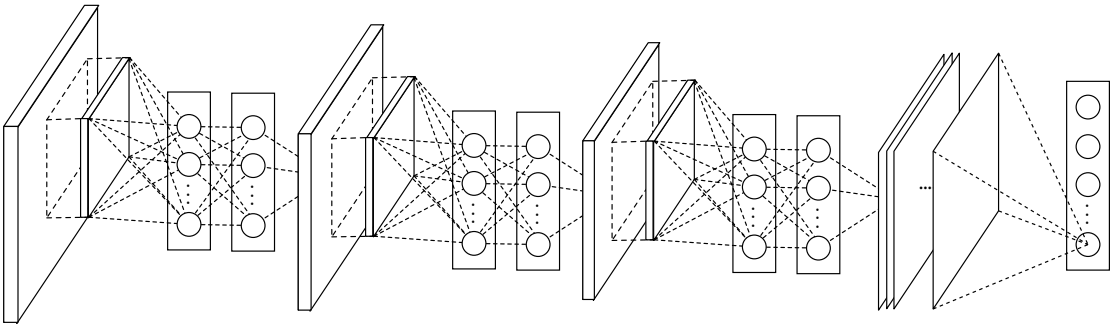


FIGURE 6.2: Network In Network architecture structure

Both the linear convolutional layer and the mlpconv layer map the local receptive field to an output feature vector. The mlpconv maps the input local patch to the output feature vector with a multilayer perceptron (MLP) consisting of multiple fully connected layers with nonlinear activation functions. The feature maps are obtained by sliding the MLP over the input in a similar manner as CNN and are then fed into the next layer.

Using these mlpconv layer, the NIN architecture advantages are:

- smaller in size: around 28 MB vs. 217 MB of AlexNet
- faster

On the other hand, NIN works well on the INRIA dataset, but taking the same model and using it to detect pedestrian on the Caltech Pedestrian Dataset, leads to poor results (see Figure 6.3).

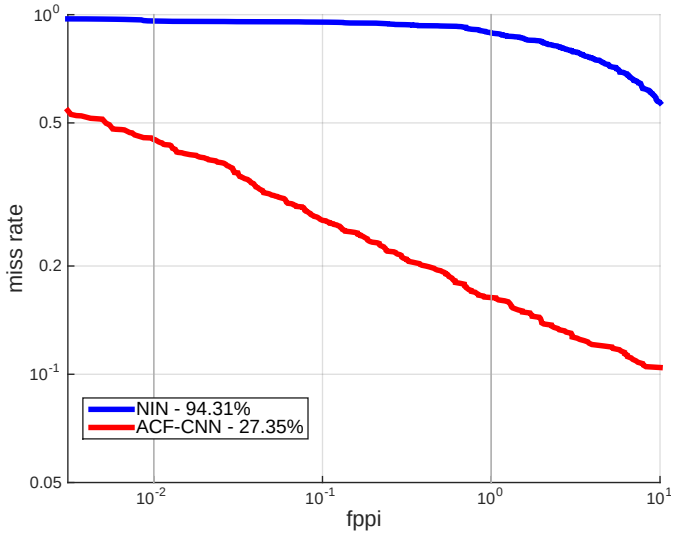


FIGURE 6.3: Network in Network ROC: comparison with the Acf-CNN model

6.2 Binarization

Since the NIN model does not perform well in terms of accuracy, the idea is to modify the AlexNet model, trying to reduce it from the $\sim$ 217 MB for storing weights, to a more affordable size. The first test in this direction is the so called "binarization" [36].

Given the weight matrix W, the sign of the element in the matrix is used in place of the weights

$$\hat{W}_{ij} = \begin{cases} 1 & \text{if} \quad W_{ij} \geq 0 \\ -1 & \text{if} \quad W_{ij} < 0 \end{cases} \tag{6.1}$$

The used threshold is identified by [36] and applied in the same way to our method.

Binarization is inspired by **Dropconnect** (see Sec 1.5.3), which randomly sets part of the parameters to 0 during training. By doing this kind of operation, the weights are compressed by a factor of 32: each weights, represented in single value precision, is now represented only by one bit.

The results of applying binarization to the AlexNet model are visible in Figure 6.4. As it's possible to see, it looses $\sim 15\%$ of log average miss rate with respect to the original model, which is not a bad result considering the compression factor.

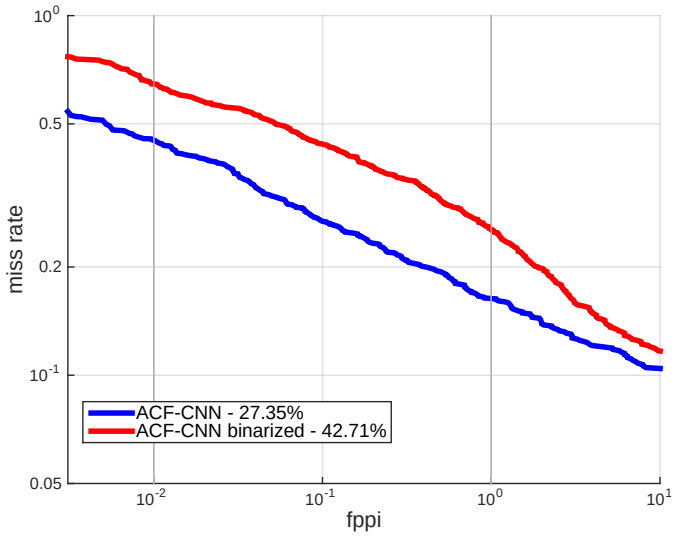


FIGURE 6.4: Binarization: ROC comparing results after training the SVM classifier

Training the SVM classifier on the features computed using the binarized model, makes the results worse. Having worse result after training the SVM classifier is an unexpected results. The reason is found in the features computation: having $\{-1, 1\}$ weights computed with $threshold = 0$, leads to highly dense features in the features space, making hard to find the line that best split the data according to the right class. Investigating more on the SVM parameters would lead to better results.

6.3 Quantization

Binarization has shown good results, with a relatively small loss compared to the compression factor. This model is however less accurate than needed, therefore a further investigation is done in order to find a model with a smaller compression factor but with acceptable accuracy.

6.3.1 K-means

K-means is an unsupervised learning algorithm of vector quantization used for cluster analysis in data mining. It aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results is a partitioning of the data space into regions. An example can be found in Figure 6.5.

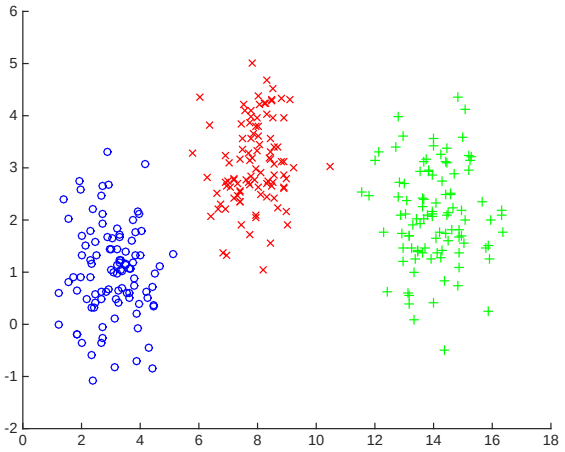


FIGURE 6.5: Quantization: k-means clustering example with two features.

The pro of k-means algorithms is that it's very eay but in the other hand, it depends a lot by the initial centroids' position, producing different results for different initializations.

For a weight matrix $W \in \mathbb{R}^{m \times n}$, all its scalar values are collected as $w \in \mathbb{R}^{1 \times mn}$, and k-means clustering is performed to the values such that:

$$\min \sum_{i}^{mn} \sum_{j}^{k} ||w_i - c_j||^2 \tag{6.2}$$

where w and c are both scalars.

Since this is a scalar quantization, this can be seen like a line where points are grouped by n elements with n the number of centroids (see Figure 6.6).

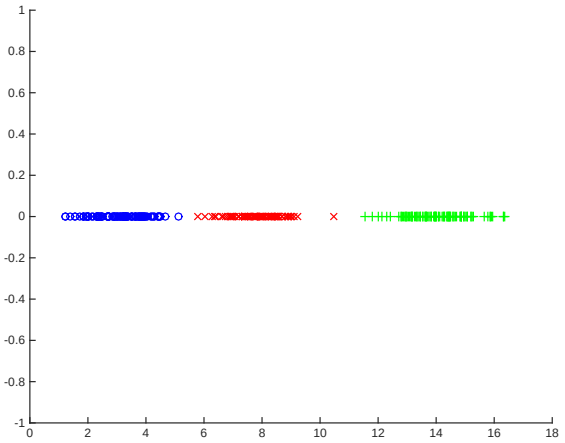


FIGURE 6.6: Quantization: k-means scalar clustering of elements

After clustering, each value in w is assigned a cluster index, and a codebook $c^{1\times k}$ is created with the cluster centers.

During prediction, by looking-up the values for each centroid in the codebook, the reconstructed matrix becomes

$$\hat{W}_{ij} = c_z \quad \text{where} \quad \min_z ||W_{ij} - c_z||^2 \tag{6.3}$$

For this approach, we need only to store indices and the codebook as parameters. Given k centers, only $\log_2(k)$ bits to encode the centers are needed.

Despite the simplicity of this approach, the experiments show that this approach gives surprisingly good performance. In Figure 6.7 the quantization using 16 centroids shows how good this method performs. This allows to compress the model by a factor

$$\frac{32\text{bits}}{\log_2(16)\text{bits}} = 8 \tag{6.4}$$

By doing this compression, we gain in term of log average miss rate, rather then loosing precision. Reason of this unexpected result is the **overfitting**: the original finetuned

model suffers overfitting, which is not visible on the validation set. When the weights are changed, the model becomes more general, resulting in better performance on the test set.

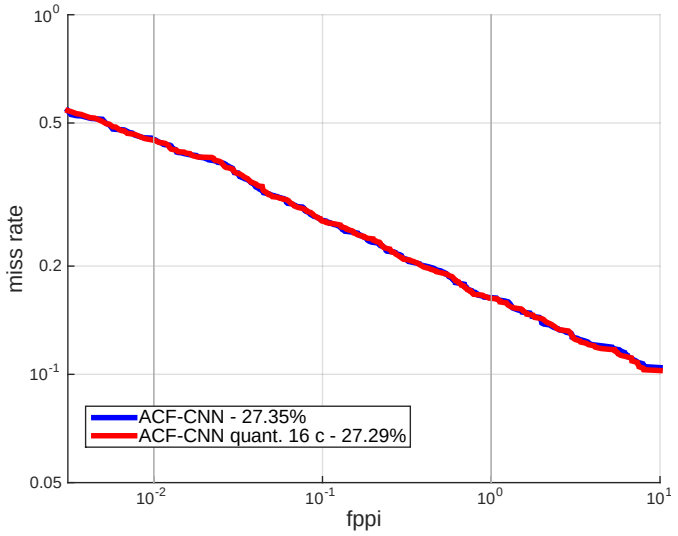
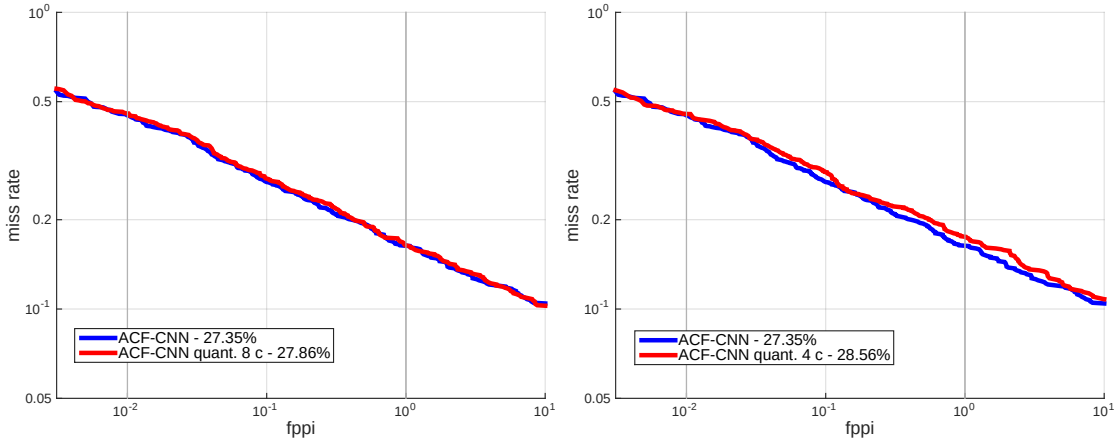


FIGURE 6.7: CNN quantization using 16 centroids

Moving to 8 centroids, hence $\frac{32}{3} = 10.667$ compression factor, the model starts loosing in performance, but still in a contained way. Its behaviour is shown together with the model compressed using 4 centroids in Figure 6.8.



CNN quantization using 8 centroids CNN quantization using 4 centroids

FIGURE 6.8: CNN quantization using 8 and 4 centroids

Model compressed using 4 centroids (16 compression factor) is a really good tradeoff between compression and loss (1.2% of log average miss rate) and will be further inspected after this analysis.

Finally, the 2 centroids (32 compression factor) is shown in Figure 6.9. Observe that this produces results are better than binarization (where split is performed according to threshold 0). Computing the mean values of the weights in each fully connected layer:

- **layer 6**: mean value $-3.331 * 10^{-4}$, standard deviation value 0.0047
- **layer 7**: mean value $-8.39 * 10^{-4}$, standard deviation value 0.0066

therefore 0 could be too coarse as threshold. Furthermore, clustering with 2 centroids does not necessary pick the mean value (also for a problem of convergency of the algorithm), and this could probably be the reason of the better result.

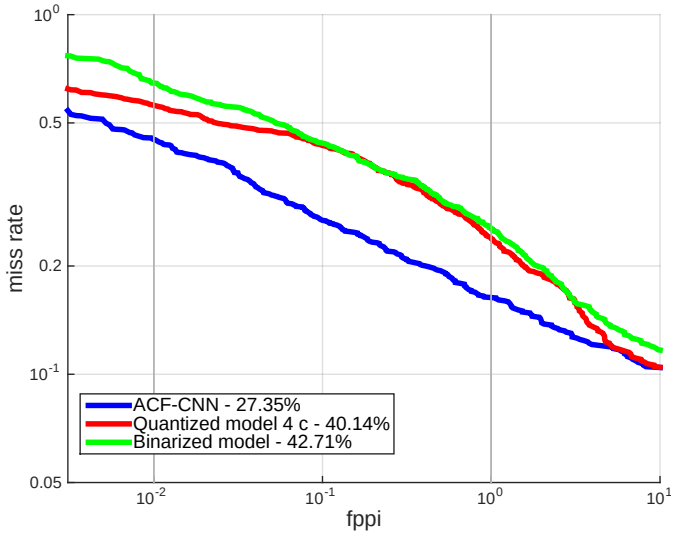
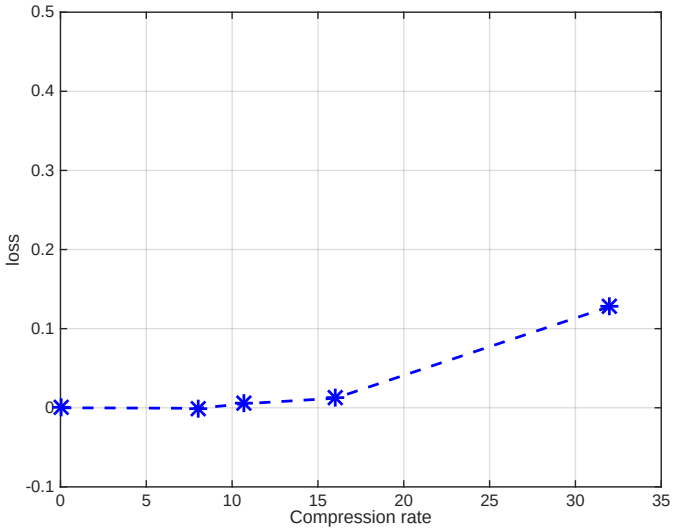


FIGURE 6.9: CNN quantization using 2 centroids

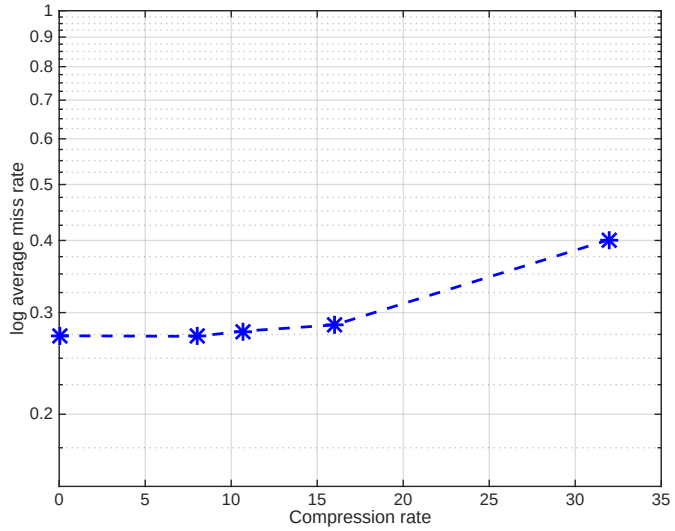
All this can be summarized by looking at figure 6.10.

6.3.2 Finetuning 4 centroids model

Taking the model compressed using 4 centroids (16 compression factor), finetuning is performed in order to find a better model that reduces the loss due to the compression.



% of log average miss rate that is lost by using a compression factor c



log average miss rate value computed using the model with a certain compression factor c

FIGURE 6.10: Compression factor vs. loss

Since the starting model is already a good one, trained on the same data, a convergence in few iterations is expected. Therefore, the parameters are chosen as follows

- base learning rate = 0.0001
- max iterations = 10000
- learning rate layers 6 and 7 = 0

Setting 0 the learning rate, the weights do not change when backpropagation is applied. The results of applying this are extremely positive: taking the iteration 3000, the resulting model produces better results than the original model (see Figure 6.11).

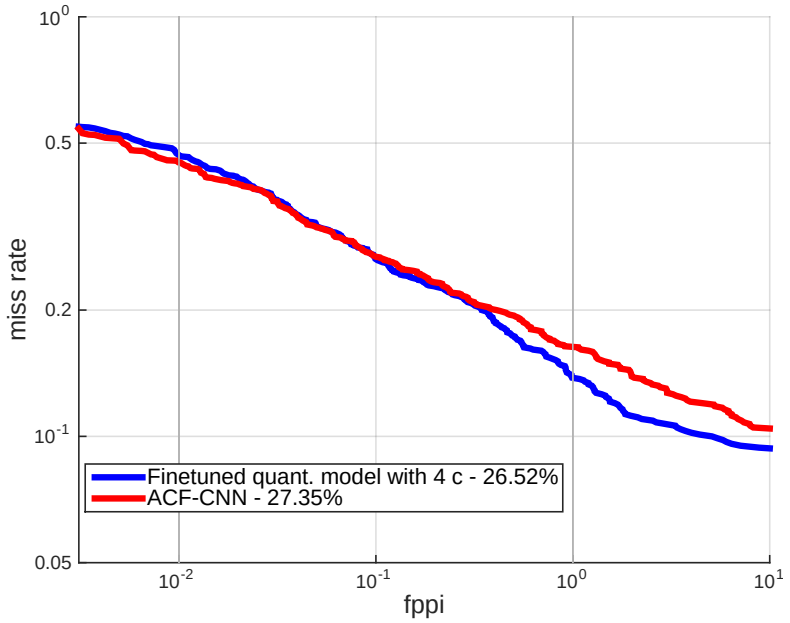


FIGURE 6.11: Finetuned quantized model with 4 centroids

Finally, applying the same approach defined in LDCF about thresholding, the result is shown in Figure 6.12.

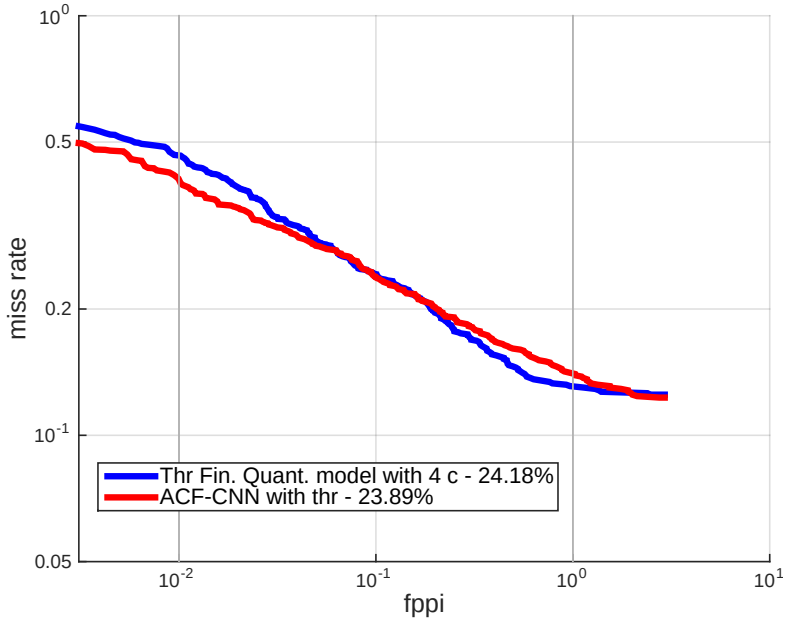


FIGURE 6.12: Finetuned quantized model with 4 centroids and thresholding

Computing the performance according the ST Microelectronics' standards, it possible to see how CNN improves the results using ST's ACF detector as selector (see Figure 6.13). In the working point 0.1 FPPI of interest, the different is

- **ACF-CNN**: 0.823
- **ACF**: 0.7451

with a 0.078 (**7.8**%) accuracy improvement.

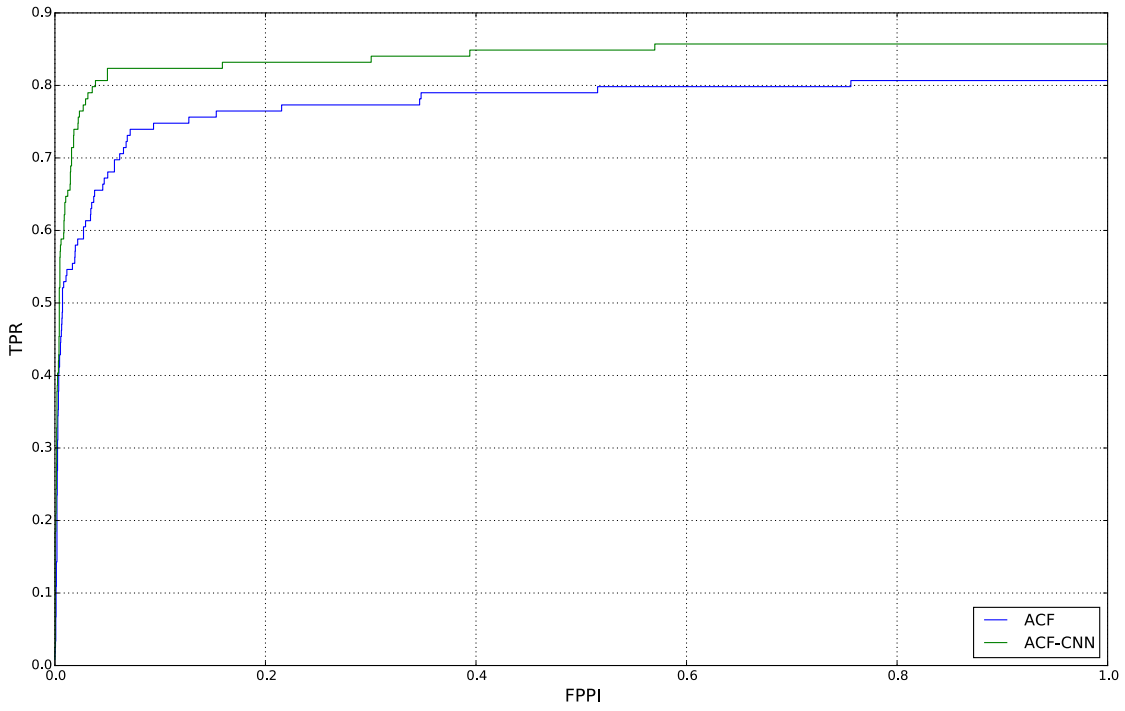


FIGURE 6.13: ACF vs. ACF-CNN according to STMicroelectronics standards

Chapter 7

Conclusions and future work

The method proposed in this work shows how Convolutional Neural Networks can be used to improve the accuracy, getting better results than the state-of-the-art methods, by dealing also with their side effect: *the size*. An interesting continuation on this line-up would be the application of other algorithms, like other clustering methods, hashing, etc. to study if it is possible to achieve better results with an higher compression factor.

With the current CNN model size, the Analyze Then Compressed (ATC) paradigm really becomes feasible, and an analysis comparing it with the other Compress Then Analyze (CTA) paradigm should be done. This in order to check if, even with large neural network as AlexNet, there is an advantage in terms of power consumption and network bandwidth in using ATC rather than CTA, as shown in [37].

Furthermore, it would be nice to test the system performance with some hybrid versions between CTA and ATC at different levels, e.g. when

- the LDCF selector is executed on the node, the data are sent through the network to the sink, where the CNN model lies and which processes them to get the results.
- the LDCF selector, plus the first five layers of the CNN model are on the node, the intermediate features are sent to the sink, where the last two more demanding layers left of the CNN compute the final results.

- ecc

Finally, based on these results, one could extend the test by checking which is the loss in terms of accuracy, if the intermediate compressed features, sent from the node to the sink, are fed to the left layers of the CNN in place of their uncompressed version, hence saving time to uncompress them.

Bibliography

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

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

[3] Evan Shelhamer. Deep learning for computer vision with caffe and cudnn. http://devblogs.nvidia.com/parallelforall/deep-learning-computer-vision-caffe-cudnn/, 2014.

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

[5] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014*, pages 818–833. Springer, 2014.

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

[7] Piotr Dollar, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: An evaluation of the state of the art. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(4):743–1761, 2012.

[8] Dariu M Gavrila and Stefan Munder. Multi-cue pedestrian detection and tracking from a moving vehicle. *International journal of computer vision*, 73(1):41–59, 2007.

[9] David Gerónimo, A Sappa, Antonio López, and Daniel Ponsa. Adaptive image sampling and windows classification for on-board pedestrian detection. In *Proceedings of the International Conference on Computer Vision Systems, Bielefeld, Germany*, volume 39, 2007.

[10] Amnon Shashua, Yoram Gdalyahu, and Gaby Hayun. Pedestrian detection for driving assistance systems: Single-frame classification and system level performance. In *Intelligent Vehicles Symposium, 2004 IEEE*, pages 1–6. IEEE, 2004.

[11] Constantine Papageorgiou and Tomaso Poggio. Trainable pedestrian detection. In *Image Processing, 1999. ICIP 99. Proceedings. 1999 International Conference on*, volume 4, pages 35–39. IEEE, 1999.

[12] Bo Wu and Ram Nevatia. Detection of multiple, partially occluded humans in a single image by bayesian combination of edgelet part detectors. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 1, pages 90–97. IEEE, 2005.

[13] Anuj Mohan, Constantine Papageorgiou, and Tomaso Poggio. Example-based object detection in images by components. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(4):349–361, 2001.

[14] Krystian Mikolajczyk, Cordelia Schmid, and Andrew Zisserman. Human detection based on a probabilistic assembly of robust part detectors. In *Computer Vision-ECCV 2004*, pages 69–82. Springer, 2004.

[15] Pedro F Felzenszwalb. Learning models for object recognition. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–1056. IEEE, 2001.

[16] Dariu M Gavrila. Multi-feature hierarchical template matching using distance transforms. In *Pattern Recognition, 1998. Proceedings. Fourteenth International Conference on*, volume 1, pages 439–444. IEEE, 1998.

[17] Dariu M Gavrila. Pedestrian detection from a moving vehicle. In *Computer Vision—ECCV 2000*, pages 37–49. Springer, 2000.

[18] Paul Viola, Michael J Jones, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. *International Journal of Computer Vision*, 63(2):153–161, 2005.

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

[20] Pierre Sermanet, Koray Kavukcuoglu, Sandhya Chintala, and Yann LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 3626–3633. IEEE, 2013.

[21] Wanli Ouyang and Xiaogang Wang. Joint deep learning for pedestrian detection. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 2056–2063. IEEE, 2013.

[22] Andrian Marcus and Oge Marques. An eye on visual sensor networks. *Potentials, IEEE*, 31(2):38–43, 2012.

[23] Alessandro Redondi, Luca Baroffio, Matteo Cesana, and Marco Tagliasacchi. Compress-then-analyze vs. analyze-then-compress: Two paradigms for image analysis in visual sensor networks. In *Multimedia Signal Processing (MMSP), 2013 IEEE 15th International Workshop on*, pages 278–282. IEEE, 2013.

[24] Clément Farabet, Berin Martini, Polina Akselrod, Selçuk Talay, Yann LeCun, and Eugenio Culurciello. Hardware accelerated convolutional neural networks for synthetic vision systems. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 257–260. IEEE, 2010.

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

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

[27] Woonhyun Nam, Piotr Dollár, and Joon Hee Han. Local decorrelation for improved pedestrian detection. In *Advances in Neural Information Processing Systems*, pages 424–432, 2014.

[28] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The elements of statistical learnin, 2009.

[29] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[30] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 580–587. IEEE, 2014.

[31] Jasper RR Uijlings, Koen EA van de Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of Computer Vision*, 104(2):154–171, 2013.

[32] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International journal of Computer Vision*, 59(2):167–181, 2014.

[33] Jan Hosang, Mohamed Omran, Rodrigo Benenson, and Bernt Schiele. Taking a deeper look at pedestrians. *arXiv preprint arXiv:1501.05790*, 2015.

[34] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[35] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. In *International Conference on Learning Representations*, 2014.

[36] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[37] Luca Bondi. Face recognition with convolutional neural networks on low power architectures. Master's thesis, Politecnico di Milano, 2014.