Object Classification and Detection in High Dimensional Feature Space

THIS IS A TEMPORARY TITLE PAGE
It will be replaced for the final print by a version
provided by the service académique.

Thèse n. 6043
présentée le 17 Décembre 2013
à la Faculté Sciences et Techniques de l'Ingénieur
Laboratoire de l'Idiap
Programme doctoral en Informatique, Communications et Information
École Polytechnique Fédérale de Lausanne

pour l'obtention du grade de Docteur ès Sciences
par

Charles Dubout

acceptée sur proposition du jury:

Prof Mark Pauly, président du jury
Dr François Fleuret, directeur de thèse
Prof Pascal Fua, rapporteur
Prof Gilles Blanchard, rapporteur
Prof Frédéric Jurie, rapporteur

Lausanne, EPFL, 2013

Acknowledgements

First and foremost, I would like to thank my thesis advisor, Dr. François Fleuret, for giving me the opportunity to carry out this very exciting project. François has all the qualities a PhD student can dream of: he is always there when you need him, ready to discuss new ideas, he has a deep knowledge not only of the field but also of CS and math in general, and he never urges you but instead gives you all the freedom you might need to carry out your work to fruition. He also taught me what it means to be a real scientist, in addition of an engineer, which I always wanted to be.

I would like also to thank my three jury members, Prof. Pascal Fua, Prof. Gilles Blanchard, and Prof. Frédéric Jurie, as well as my jury president, Prof. Mark Pauly, for doing me the honor to supervise my oral exam.

Many thanks to Dr. Raghuraman Krishnamoorthi, Dr. Bojan Vrcelj, and all their colleagues for giving me the opportunity to come to the U.S. and mentoring me during my internship at Qualcomm San Diego. Experiencing life in California and working in a large IT company was very interesting and something I always wanted to try.

Working at Idiap would not have been as fun without all my office mates and colleagues: Adolfo, Alexandre[2], Alexandros, André, Arjan, Ashtosh, Barbara, Bastien, Cheng, Chidansh, Chris, Cosmin, Daira, David, Elie, Flavio, Francesco, Gulcan, Gwénolé, Hugo, Hui, Ilja, Ivana, Jagan, James, Jean-Marc, Joan, Joel, Kenneth, Laurent[2], Leo, Majid, Manuel, Marc, Marco, Maryam, Mathew, Minh-Tri, Nadine, Nesli, Nicolae, Nik, Nikolaos, Novi, Olivier, Oya, Paco, Paul, Philip[2], Pierre-Edouard, Radu, Raphaël, Riwal, Roger, Romain, Ronan, Roy, Rui, Rémi[2], Samira, Serena, Sylvie, Tatiana, Teodora, Thomas, Valérie, Vincent, Yann and probably a few others which I forgot, sorry! I will not soon forget all the long baby foot games we played, all the Friday afternoon beers, and all our (sometimes a bit pointless, Boosting vs. SVM anyone?) discussions.

I would finally like to thank my parents, in-laws, brothers and sisters, and all the rest of my family for their constant support. Special thanks to my wife Wenqi, always there to share with me the highs and lows of PhD life and without who none of it would have been possible.

Charles Dubout was supported by the Swiss National Science Foundation under grant 200021-124822 – VELASH.

Lausanne, 20 December 2013 Charles

[2] Both of you.

Abstract

Object classification and detection aim at recognizing and localizing objects in real-world images. They are fundamental computer vision problems and a prerequisite for full scene understanding. Their difficulty lies in the large number of possible object positions and the appearance variations of object classes. This thesis improves upon several classical machine learning algorithms, enabling large computational gains in high dimensional feature space.

A common trend in machine learning and computer vision research is to go large scale. In particular, the advent of huge datasets mined from the Internet, and the combination of multiple feature sources have considerably broadened the applications of computer vision. Tasks which were thought impossible a few years ago, such as human action recognition or pose estimation, automatic outdoor navigation, *etc.*, now seem within reach.

This dissertation is divided into two parts. The first one deals with the efficient training of a classifier or detector based on a large number of feature extractors, outside the control of the learning algorithm, and therefore of unknown suitability to the task at hand. More precisely, this part presents two kinds of strategies to accelerate the training of Boosting algorithms in such a context: (a) a method to better deal with the increasingly common case where features come from multiple sources (*e.g.* color, shape, texture, *etc.*, in the case of images) and therefore can be partitioned into meaningful subsets; (b) new algorithms which balance at every Boosting iteration the number of weak learners and the number of training examples to look at in order to maximize the expected loss reduction. Experiments in image classification and object recognition on four standard computer vision datasets show that the adaptive techniques we propose outperform both basic sampling and state-of-the-art bandit methods.

The second part deals with linear object detectors, currently the most popular class of detection systems, encompassing template matching, deformable part models, poselets, convolutional neural networks (which internally use linear filters), *etc.* The main bottleneck of many of those systems is the computational cost of the convolutions between the multiple rescalings of the image to process and the linear filters. We make use of properties of the Fourier transform and clever implementation strategies to obtain a speedup factor proportional to the filter size, both while training and at test time. We also introduce a few modifications to the original Deformable Part Model (DPM) of Felzenszwalb *et al.* improving its detection accuracy. The gains in performance are demonstrated on the well-known Pascal VOC benchmark, where an increase by one order of magnitude in the speed of said convolutions, and an average improvement of 15% in the accuracy of the detector are established.

Keywords: Boosting, large scale learning, feature selection, linear object detection, deformable part model

Résumé

La classification et la détection d'objets visent à reconnaître et à localiser des objets dans des images du monde réel. Ce sont des problèmes fondamentaux de vision par ordinateur qui constituent un prérequis à la compréhension de scènes complètes. Leur difficulté vient du large nombre de positions potentielles et de la diversité d'apparence propres à chaque classe d'objets. Cette thèse présente plusieurs améliorations d'algorithmes classiques d'apprentissage automatique, diminuant grandement leur coût computationnel en espaces de caractéristiques de grandes dimensions.

Une des tendances actuelles de la recherche en apprentissage automatique et en vision par ordinateur est de considérer des échelles toujours plus grandes. En particulier, l'avènement d'énormes ensembles de données compilés à partir d'Internet, et la combinaison de plusieurs sources de caractéristiques visuelles ont considérablement étendu les champs d'application de la vision par ordinateur. Des tâches qui semblaient impossible il y a quelques années, telles que la reconnaissance de l'activité ou de la pose d'êtres humains, la navigation automatique en extérieurs, *etc.*, semblent maintenant proches d'être réalisables.

Cette dissertation est divisée en deux parties. La première traite de l'entraînement efficace d'un classificateur ou d'un détecteur basé sur un grand nombre d'extracteurs de caractéristiques visuelles, hors du contrôle de l'algorithme d'apprentissage, et dont la pertinence vis-à-vis de la tâche à résoudre est inconnue. Plus précisément, cette première partie présente deux types de stratégies visant à accélérer l'entraînement d'algorithmes de Boosting dans ce contexte : (a) une méthode pour gérer le cas de plus en plus courant où les caractéristiques sont issues de plusieurs sources (*ex.* couleur, forme, texture, *etc.*, dans le cas d'images) et peuvent donc être partitionnées en sous-ensembles de façon non-arbitraire ; (b) de nouveaux algorithmes qui équilibrent à chaque itération de Boosting le nombre de classifieurs faibles et le nombre d'exemples d'apprentissage dans le but de maximiser l'espérance de la réduction de la fonction de coût. Quatre expériences en classification d'images et en reconnaissance d'objets sur des ensembles de données standards montrent que les techniques adaptatives que nous proposons surpassent des techniques d'échantillonnage basiques ainsi que des méthodes de pointe utilisant des bandits manchots.

La seconde partie traite de détecteurs d'objets linéaires, actuellement la classe de détecteurs la plus populaire, incluant la comparaison avec des motifs standards, les modèles à parties déformables, les poselets, les réseaux de neurones à convolution (qui utilisent des filtres linéaires en interne), *etc.* Le principal goulot d'étranglement de la plupart de ces systèmes est

le coût computationnel des convolutions entre les multiples redimensionnements de l'image à traiter et des filtres linéaires. En utilisant certaines propriétés de la transformée de Fourier ainsi que d'ingénieuses stratégies d'implémentation, nous obtenons un gain d'accélération proportionnel à la taille des filtres, à la fois durant l'entraînement et durant le test. Nous présentons aussi quelques modifications apportées au modèle à parties déformables originel de Felzenszwalb *et al.* améliorant sa précision en détection. Les gains apportés en performance sont démontrés sur le célèbre Pascal VOC benchmark. Une accélération de la vitesse des convolutions d'un ordre de grandeur, ainsi qu'une amélioration moyenne de la précision du détecteur de 15% sont démontrées.

Mots-clés : Boosting, apprentissage à grande échelle, sélection de caractéristiques, détection d'objet linéaire, modèle à parties déformables

Contents

List of Figures

List of Algorithms

List of Tables

1 Introduction

This introduction presents an overview of learning in high dimensional feature space (§ 1.1) and the contribution of this thesis (§ 1.2). The motivations presented here are elaborated further in the following chapters. At the end of this chapter, we also introduce the notation and the necessary mathematical tools used in this thesis § 1.3.

1.1 Learning in High Dimensional Feature Space: Advantages and Challenges

A common trend in machine learning and computer vision research is to go "large scale", both in the number of training examples (*e.g.* ImageNet (Deng et al., 2009) contains currently close to ten million images, i.e. approximately ten terabyte of raw data) and the number of features considered (*e.g.* shape, color, texture, *etc.*). Such increase in the size of datasets and the number of features, together with the advent of more sophisticated learning algorithms led to major improvement in classification performance and in the range of problems which can now be tackled (action recognition, pose estimation, automatic outdoor navigation, *etc.*). In particular using more features can improve performance by increasing the amount of information given to the learning system, make the system more robust, and can even make the problem simpler by making it more separable. Two practical examples are displayed in table 1.1 and in figure 1.1. The first example, adapted from (Gehler and Nowozin, 2009), reports the test accuracy of various kernel methods trained either on a single or on a combination of all the seven features used by the authors in their experiments. The main observation they made is that the learning method often does not matter much, all of the algorithms performing similarly in that particular experiment, but that using a combination of features can be crucial in order to get the best out of a classifier. The second example, adapted from (Fleuret et al., 2011), plots the test error of an AdaBoost classifier on the Synthetic Visual Reasoning Test (SVRT) for various number of training examples and groups of features. SVRT is a series of 23 image classification problems, each containing images of simple shapes, positive or negative according to some high-level rule, typically easy to understand for humans but hard for off-

Table 1.1 – Mean classification accuracy on the Oxford Flowers dataset (Nilsback and Zisserman, 2006) of several classification methods using either a single features or a combination of all of them. The learning method does not matter much, combining features is much more important. Reprinted from (Gehler and Nowozin, 2009).

Single feature			Combination methods		
Method	Accuracy	Time	Method	Accuracy	Time
Color	60.9 ± 2.1	3	product	85.5 ± 1.2	2
Shape	70.2 ± 1.3	4	averaging	84.9 ± 1.9	10
Texture	63.7 ± 2.7	3	CG-Boost	84.8 ± 2.2	1225
HOG	58.5 ± 4.5	4	MKL (SILP)	85.2 ± 1.5	97
HSV	61.3 ± 0.7	3	MKL (Simple)	85.2 ± 1.5	152
siftint	70.6 ± 1.6	4	LP-β	85.5 ± 3.0	80
siftbdy	59.4 ± 3.3	5	LP-B	85.4 ± 2.4	98

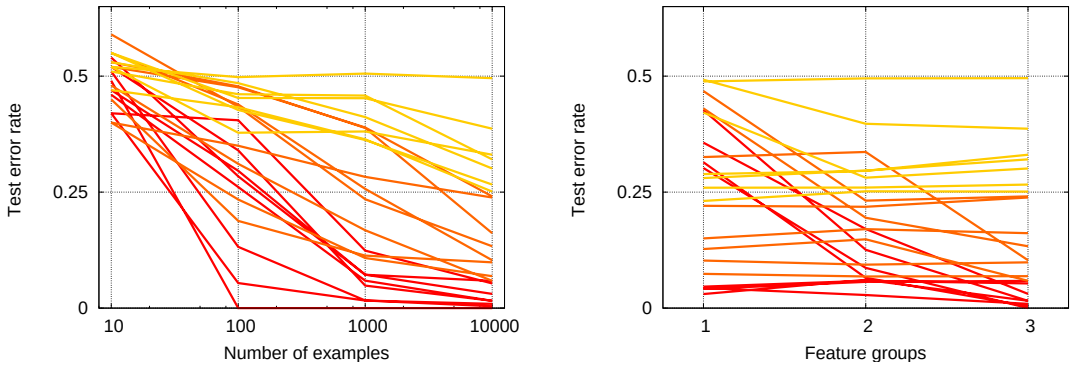
the-shelf machine learning algorithms. The features from group 1 only count pixels in boxes, those of group 2 also look at edges, and the ones in group 3 also look at properties of the whole image (Fourier and wavelet coefficients). The performance of the classifier increases with both, overfitting decreasing with the number of training examples, while the more complex features make some tasks much easier to solve. For example tasks which necessitate to match shapes become easier with features from group 2, while tasks necessitating to look at the whole image (for example to detect alignment or symmetry) become much easier with the ones from group 3. A more in-depth analysis of SVRT and these results is the topic of chapter 2.

But this trend towards larger and larger datasets also poses deep scalability issues, since it might become difficult to store and process all this information. For instance it might be impossible for an object detector to extract and store all patches from all training images of a large dataset on current hardware.

Current feature combination techniques such as classifier or feature concatenation, multiple kernel learning (MKL) (Lanckriet et al., 2004; Bach et al., 2004) or LP-β (Gehler and Nowozin, 2009) pay little attention to their computational cost, and assume that all the features have been selected by an expert, meaning that they do not expect most features to be irrelevant. Even when considering a single kind of feature, as is often the case in object detection, the amount of data that has to be processed is often huge due to the sheer number of overlapping image sub-windows, so that even 'fast' linear methods can struggle.

Our aim is therefore to address the following research questions:

- How to train a classifier efficiently using multiple kind of uncontrolled features, of various usefulness?



(a) Influence of the number of examples.

(b) Influence of the number of features.

Figure 1.1 – Influence of the number of training examples and the amount/complexity of the features on the Synthetic Visual Reasoning Test (SVRT). The features from group 1 only count pixels in boxes, those of group 2 also look at edges, and the ones in group 3 look at properties of the whole image (Fourier and wavelet coefficients). Reprinted from (Fleuret et al., 2011).

- How to make large scale learning faster (large in the number of examples and features)?
- Is it possible to speed up object detection over dense features by exploiting the overlap between samples?
- How to best exploit this speed up to improve detection accuracy?

We propose to consider the first two questions in the Boosting framework. We believe Boosting to be well suited for this task due to its iterative building process, enabling it to do feature selection while training. Each training iteration can be restricted to look only at a small subset of examples and features, limiting the total computational cost, instead of looking at everything all the time, as would be the case with support vector machines (Vapnik, 1995) or classical neural networks. Its linear nature also makes it easy to understand the contribution of each feature, and to prune away useless ones.

For the last two questions we turned to linear classifiers, which have been hugely popular in recent years in the vision community. We focus particularly on the Deformable Part Model (DPM) of Felzenszwalb *et al.* (Felzenszwalb et al., 2010b), which has received the most attention in recent years, but our analyses remain applicable to a wider range of object detectors.

1.2 Organization and Contribution of this Thesis

This thesis is organized in two parts. After defining Boosting and stressing out the importance of large scale learning in chapter 2, using SVRT as an illustration, chapter 3 describes three new families of algorithms to improve Boosting in high dimensional feature space, particularly when dealing with multiple kind of features and a large number of examples.

The first one, Tasting, is a strategy to bias feature sampling towards promising subsets. Contrarily to previously existing methods (LazyBoosting (Escudero et al., 2000), AdaBoost.UCB (Busa-Fekete and Kegl, 2009) and its later variants (Busa-Fekete and Kegl, 2010), *etc.*), it continuously estimates the expected quality of each feature subset from a limited set of features sampled prior to the learning, as well as the current Boosting weights. As for the bandit-related methods which we use as baselines, Tasting exploits the fact that the full feature set is a heterogeneous union of somehow homogeneous subsets of features. It exploits the main strength of Boosting which is to spot and combine complementary features, and can thus discard features redundant with features already chosen.

The second one, Maximum Adaptive Sampling (abbreviated M.A.S.) is a family of algorithms targeted at learning in high dimensional feature space with or without multiple feature subsets. They model at every Boosting step the distribution of the performance of the weak learners, and computes from it the optimal number of examples and weak learners to sample under a given cost constraint.

The third one, Laminating, tries to reduce the requirement for a density model of the weak learners' performance. At every Boosting step it iteratively halves the number of considered weak learners, and doubles the number of samples, until only one weak learner remains.

The second part proposes acceleration strategies applicable to a wide class of linear object detector. It focuses particularly on the currently state-of-the-art linear object detector of Felzenszwalb *et al.* (Felzenszwalb et al., 2010b), and describes improvement to its efficiency and its detection accuracy.

Current state-of-the-art linear object detection methods compute the convolutions between image features and trained linear filters directly, with a cost proportional to the size of the linear filters. They work by first extracting features from the input images, often at multiple resolutions. Those image and filter features can be seen as being organized in planes, each containing a distinct feature at the same image locations. Existing methods compute the convolution of an image feature planes and a linear filter by first convolving each feature plane independently, and summing the results together. The novelty of our invention exposed in chapter 4 is to do the convolution using a Fast Fourier Transform (FFT) algorithm and to take advantage of the linearity of the transform to reduce the number of required inverse transforms to one per filter at detection time (instead of the number of features). We also came up with two additional implementations strategies, both necessary in order to get the most out of the method, and obtain one order of magnitude speedup compared to previous implementations.

In chapter 5 we rewrite the computation of the gradient of the loss minimized during training as a convolution. This enables us to accelerate training as well for any loss written as a sum over the examples, making the overall training computational cost independent of the filters' sizes. It relieves all the constraints inherent to sparse and approximate methods, but is not always as efficient, as we experimentally observed.

Chapter 6 presents three extensions to the original DPM of Felzenszwalb *et al.* The first one is to use different kinds of features in addition to HOG. We settled for histograms of Local Binary Patterns (LBP), a widely used texture descriptors, and our own color histograms. These are similar to HOG in that they are local histograms computed over the pixels of each cell of a dense grid, but instead of being histograms of the gradient orientation weighted by the gradient magnitude, they are respectively histograms of the LBP binary code or the hue weighted by the saturation.

The second one removes a limitation of current DPMs, in which parts deform only at a fixed predetermined scale relative to that of the root of the models (typically at twice the resolution). They do so because it enables them to find the optimal placement of each part efficiently, using a fast 2D distance transform algorithm. By settling for approximately optimal placements, we were able to efficiently deform the parts across scales as well, by reusing the original convolutions and distance transforms. Allowing parts to move in 3D increases the expressivity of the models, and might approximate an increase in the scanning resolution.

The third extension addresses a shortcoming of the standard DPM: its complete ignorance of joint aspects of appearance, marginalizing it completely over the parts. We therefore proposes to add to the model a term looking jointly at the appearance of the part and the root together, which has the advantage compared to other joint models to be simpler and more efficient.

1.3 Notation

In this section we introduce formally the notations as well as the necessary mathematical tools used in this thesis.

Notation. We indicate scalars with lower case letters (*e.g.* x and λ), vectors and matrices with bold letters (*e.g.* $\mathbf{x}$ and $\mathbf{w}$), and sets with calligraphic font (*e.g.* $\mathscr{X}$ and $\mathscr{H}$). In order to make index expressions more readable, we use $x(i,\ j)$ rather than x_{ij} to refer to the element in the i^{th} row and the j^{th} column of matrix $\mathbf{x}$. Thus $x_k(i,j)$ signifies the element of indices i and j in $\mathbf{x}_k$, the matrix of index k. The set of real numbers is denoted by $\mathbb{R}$ and the indicator function, taking the value 1 if the expression inside is true and 0 otherwise is denoted by $\mathbf{1}_{\{\cdot\}}$.

Part I Boosting in High Dimensional Feature Space

2 Influence of the number of Training Examples and Features on Boosting

In this chapter we present our observations on the influence of the number of training examples and features on the classification performance of AdaBoost, the most popular Boosting algorithm. The dataset that we used is the Synthetic Visual Reasoning Test (SVRT), a collection of twenty-three synthetic image classification problems. While only a few images are necessary for humans to grasp the rule underlying each problem, general machine learning methods often require thousands of examples as well as elaborate image features to achieve their optimal performance. Content presented in this chapter is based on the following publications (Fleuret et al., 2011):

> F. Fleuret., T. Li, C. Dubout, E. K. Wampler, S. Yantis, and D. Geman. Comparing machines and humans on a visual categorization test. In *Proceedings of the National Academy of Sciences*, 2011.

2.1 Introduction and related works

Boosting is a powerful approach to improve the performance of a given "weak" learning algorithm (*i.e.* one that performs just slightly better than random guessing) by combining them into a "strong" learning algorithm. While Boosting is not algorithmically constrained, most Boosting algorithms work by iteratively training the same weak classifier with a different weighting over the training examples. At each iteration, the weighting distribution gives emphasis to the "hardest" (most incorrectly classified) examples. The final "strong" classifier is obtained as an average of the trained weak learners, weighted by some function of their respective accuracy. Under some mild assumptions, and given a sufficient number of iterations, the training error of the final combination can become arbitrarily low (Schapire et al., 1998). It has also been repeatedly observed in practice that Boosting is relatively immune to overfitting, as the testing error typically continues to decrease long after the training error reaches zero.

2.1.1 AdaBoost

In this thesis we will focus on the (discrete) AdaBoost (short for Adaptive Boosting) algorithm, by far the most popular and extensively studied Boosting algorithm (Friedman et al., 2000). We concentrate on the binary classification task and let the training set be

$$(\mathbf{x}_n, y_n) \in \mathcal{X} \times \{-1, 1\},\ n = 1, \ldots, N. \tag{2.1}$$

The goal is to construct a strong classifier of the form

$$F(\mathbf{x}) = \operatorname{sign}\big(f(\mathbf{x})\big) \tag{2.2}$$

with

$$f(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) \tag{2.3}$$

where $h_t(\mathbf{x})$ denotes a binary weak learner, *i.e.* a function of the form $\mathcal{X} \to \{-1, 1\}$ and $\alpha_t \in \mathbb{R}$ denotes its weight.

Given a set of weak learners $\mathcal{H}$, the choice of $h_t \in \mathcal{H}$ at each iteration results from the minimization of the Exponential loss

$$L(f) = \sum_{n=1}^{N} \exp\big(-y_n f(\mathbf{x}_n)\big). \tag{2.4}$$

This cost function penalizes samples that are wrongly classified ($y_n f(\mathbf{x}_n) \leq 0$) much more heavily than those that are classified correctly ($y_n f(\mathbf{x}_n) > 0$). It upper-bounds the Hamming loss, which in the binary case is equivalent to the training error (see figure 2.1). Directly optimizing (2.4) is complex and AdaBoost employs instead a greedy approach, optimizing each pair of α_t, h_t iteratively (see algorithm 2.1).

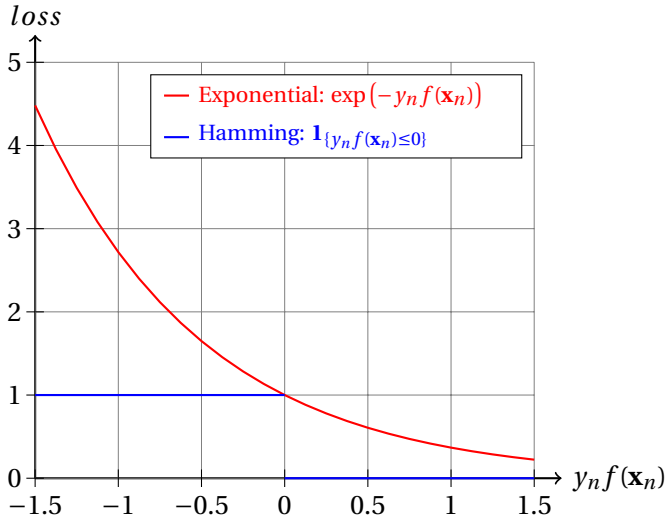


Figure 2.1 – An illustration of how the Exponential loss (in red) upper-bounds the Hamming loss (in blue), directly related to the training error since $\mathbf{1}_{\{F(\mathbf{x}_n)\neq y_n\}} = \mathbf{1}_{\{y_n f(\mathbf{x}_n)\leq 0\}}$.

Suppose that t pairs of α_t, h_t have already been optimized, and let

$$f_t = \sum_{i=1}^{t} \alpha_i h_i. \tag{2.5}$$

stands for the current classifier. The next pair picked by AdaBoost is

$$(\alpha_{t+1}, h_{t+1}) = \underset{(\alpha\in\mathbb{R},\, h\in\mathscr{H})}{\operatorname{argmin}} L(f_t + \alpha h) \tag{2.6}$$

$$= \underset{(\alpha\in\mathbb{R},\, h\in\mathscr{H})}{\operatorname{argmin}} \sum_{n=1}^{N} \exp\big(-y_n\big(f_t(\mathbf{x}_n) + \alpha h(\mathbf{x}_n)\big)\big) \tag{2.7}$$

$$= \underset{(\alpha\in\mathbb{R},\, h\in\mathscr{H})}{\operatorname{argmin}} \sum_{n=1}^{N} \exp\big(-y_n f_t(\mathbf{x}_n)\big) \exp\big(-y_n \alpha h(\mathbf{x}_n)\big). \tag{2.8}$$

If we define

$$\omega_t(n) = \frac{\exp\big(-y_n f_t(\mathbf{x}_n)\big)}{\sum_{i=1}^{N} \exp\big(-y_i f_t(\mathbf{x}_i)\big)} = \frac{\exp\big(-y_n f_t(\mathbf{x}_n)\big)}{L(f_t)} \tag{2.9}$$

the normalized weight associated with example n, we can rewrite (2.8) as

$$(\alpha_{t+1}, h_{t+1}) = \underset{(\alpha\in\mathbb{R},\, h\in\mathscr{H})}{\operatorname{argmin}} \sum_{n=1}^{N} \omega_t(n) \exp\big(-y_n \alpha h(\mathbf{x}_n)\big). \tag{2.10}$$

Algorithm 2.1 AdaBoost, the most common Boosting algorithm.

Input: $(\mathbf{x}_n, y_n) \in \mathcal{X} \times \{-1, 1\}$, $n = 1, \ldots, N$, $\mathcal{H}$, T
  for $n \leftarrow 1, \ldots, N$ **do**
    $\omega_1(n) = \frac{1}{N}$ `# Set the Boosting weights uniformly`
  end for
  for $t \leftarrow 1, \ldots, T$ **do**
    $h_t \leftarrow \underset{h \in \mathcal{H}}{\operatorname{argmax}} \epsilon_t(h)$, where $\epsilon_t(h) = \sum_{n=1}^{N} \omega_t(n) y_n h(\mathbf{x}_n)$ `# Find the best weak learner`
    $\alpha_t \leftarrow \frac{1}{2} \log\left(\frac{1+\epsilon_t(h_t)}{1-\epsilon_t(h_t)}\right)$ `# Optimal weak learner weight`
    for $n \leftarrow 1, \ldots, N$ **do**
      $\omega_{t+1}(n) = \frac{\omega_t(n) \exp\left(-y_n \alpha_t h_t(\mathbf{x}_n)\right)}{\sum_{i=1}^{N} \omega_t(i) \exp\left(-y_i \alpha_t h_t(\mathbf{x}_i)\right)}$ `# Update the Boosting weights`
    end for
  end for
Output: $f = \sum_{t=1}^{T} \alpha_t h_t$ `# Return the strong classifier`

It is now easy to see that the solution of (2.10) is

$$h_{t+1} = \underset{h \in \mathcal{H}}{\operatorname{argmax}} \epsilon_t(h) \tag{2.11}$$

$$\alpha_{t+1} = \frac{1}{2} \log\left(\frac{1+\epsilon_t(h_{t+1})}{1-\epsilon_t(h_{t+1})}\right). \tag{2.12}$$

where $\epsilon_t(h)$ is called the edge of weak learner h and is defined as

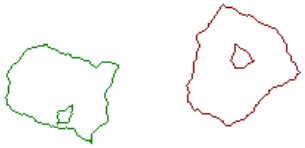
$$\epsilon_t(h) = \sum_{n=1}^{N} \omega_t(n) y_n h(\mathbf{x}_n). \tag{2.13}$$

2.2 Experiments

The Synthetic Visual Reasoning Test is a collection of twenty-three binary image classification problems. Each problem consists of two sets of images, each generated by a computer program. This ensures that the potential number of images is virtually infinite and can not be modeled properly with a brute-force memorization. The images are black and white and of resolution 128×128 pixels (see figure 2.1). The problems are designed so that the two categories can be separated without mistakes if the underlying rule is known.

The authors created seven non-exclusive families of rules: (1) parts with identical shape, differing only in size and/or orientation, (2) proximity and contact of parts, (3) intra-distance in groups of parts, (4) symmetry of (group of) parts, (5) groups of parts of specific cardinality, (6) inclusion of parts inside larger parts, and finally (7) ordering of parts along a line.

(a) Problem 1.

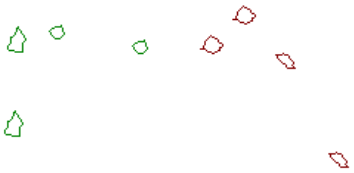


(b) Problem 2.

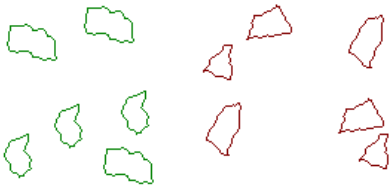
(c) Problem 3.

(d) Problem 4.

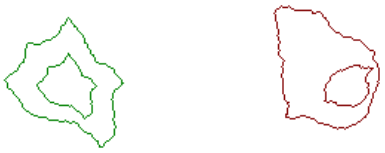
(e) Problem 5.



(f) Problem 6.

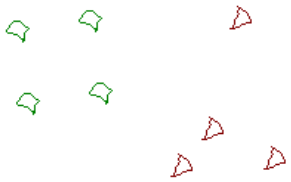


(g) Problem 7.

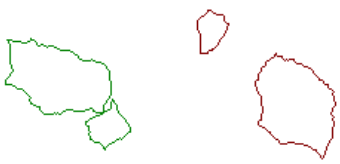


(h) Problem 8.

(i) Problem 9.

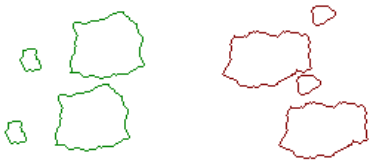


(j) Problem 10.



(k) Problem 11.

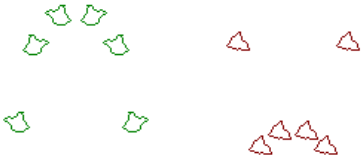
(l) Problem 12.



(m) Problem 13.

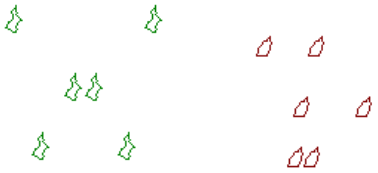
(n) Problem 14.

(o) Problem 15.



(p) Problem 16.

(q) Problem 17.

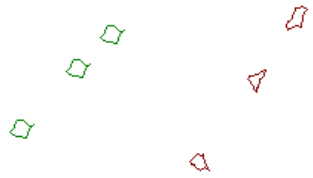


(r) Problem 18.

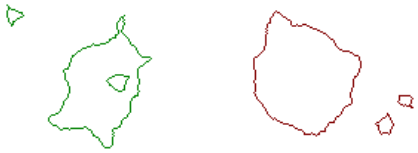
(s) Problem 19.

(t) Problem 20.

(u) Problem 21.



(v) Problem 22.



(w) Problem 23.

Figure 2.1 – A pair of positive (in green) and negative (in red) images from each of the 23 image classification problems making up the Synthetic Visual Reasoning Test (SVRT).

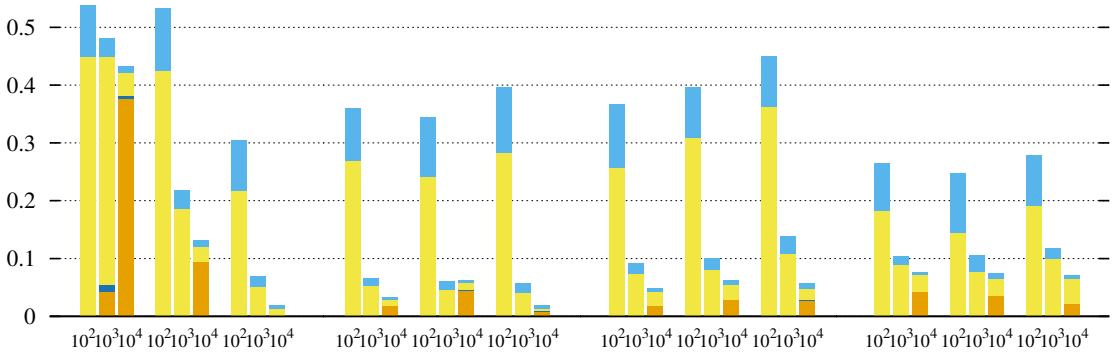
We performed experiments using the AdaBoost algorithm using decision stumps (thresholded feature; decision tree of depth 1) as weak learners with three groups of features of increasing complexity. The features of group 1 just compute the number of black pixels over rectangular areas in the image, features from group 2 are all related to the presence of edges in the image, and the features from group 3 are related to the spectral properties of the image (Fourier and wavelet coefficients). In the following we consider that feature group 2 includes group 1, and similarly that group 3 includes groups 1 and 2.

We report in figure 2.2 and in table 2.1 the test error estimated on 10,000 images of classifiers trained with different number of training examples (100, 1,000, and 10,000) and different set of features. Even though the test error vary wildly from 0% (on problem 16) to 50% (on problem 21) depending on the problem, some trends are clear.

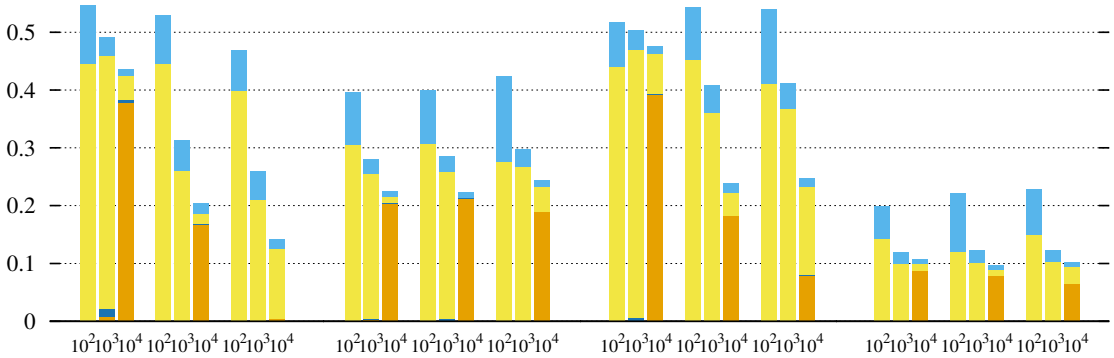
The performance of the AdaBoost classifier increases strictly with both the number of training examples and the complexity of the features. The test error decreases logarithmically with the number of training examples on all the problems, for example using all the features the mean test errors across all problems are 49.6%, 34.0%, 23.0%, and 15.8% training respectively with 10, 100, 1,000, and 10,000 examples, as can be read in the last column of table 2.1. Considering that a problem is 'solved' when its associated test error drops below 10%, no problem are solved training with only 10 examples, 2 when training with 100 examples, 10 when training with 1,000 examples, and 11 when training with 10,000 examples. The features used are also of importance, as the mean test error using the basic black pixel counting features of group 1 only drops from 25.8% to 19.1% when using also the edge related features of group 2, and to 15.8% when using also the spectral features of group 3. Although the performance does not strictly increase when using additional features (probably because of overfitting), it is very close to be the case. Using the same criteria as previously, it is this time 2 problems which can be solved using features from group 1, 8 when using group 2, and 11 using group 3.

The interpretation of these results is extremely difficult, as machine learning may rely on cues which seem at first irrelevant to the actual structure of the problem. For instance, images from problem 8 contain two closed shapes of different sizes. In positive images, the small shape is enclosed by the larger one, while in negative images they stand next to each others. While the rule involves reasoning about the spatial relations of the shapes, they can be approximately separated looking only at the distribution of the black pixels, which will be more concentrated for positive images and more spread out for negatives. Simply thresholding the variance of the pixels leads to a 9% test error, already 'solving' the problem. Symmetry with respect to a centered axis induces a more balanced repartition of the black pixels, as they can not anymore be all located on one side of the said axis, although Fourier features also seem to be able to help with the problem quite a bit.

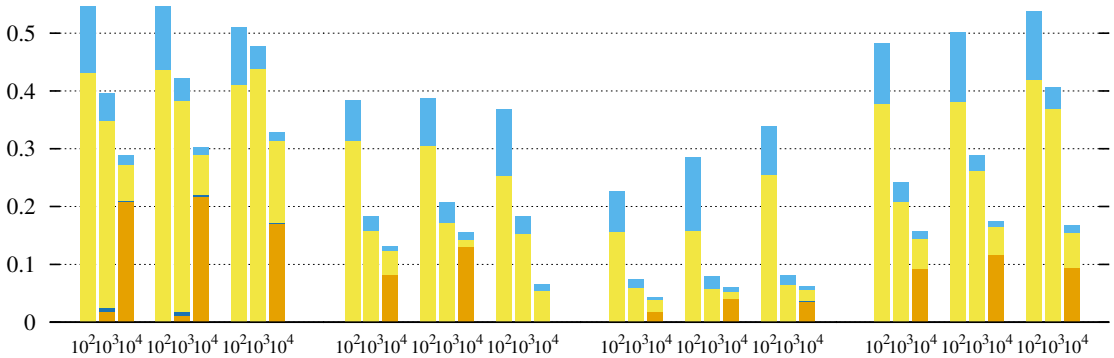
All these cues provide information about the state of interest, but have a very indirect relation to the underlying rule. In the end, a few exact geometrical properties are probably perceived by the classification algorithm through a multitude of slightly unbalanced statistical cues.



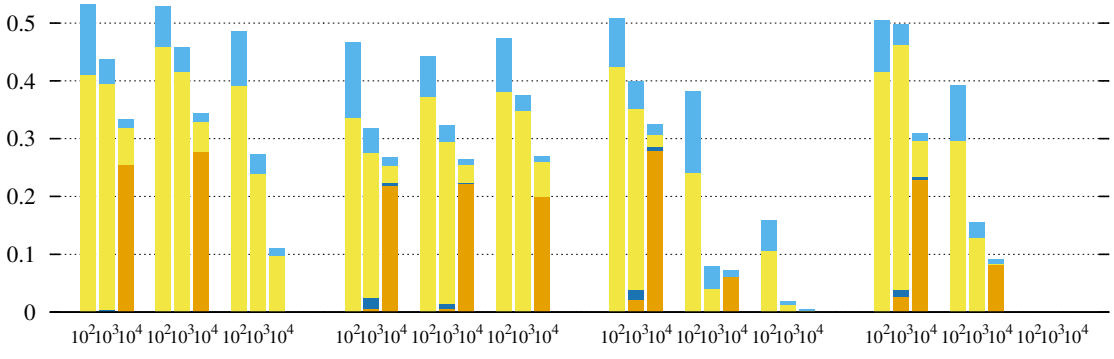
(a) Problem 1. (b) Problem 2. (c) Problem 3. (d) Problem 4.



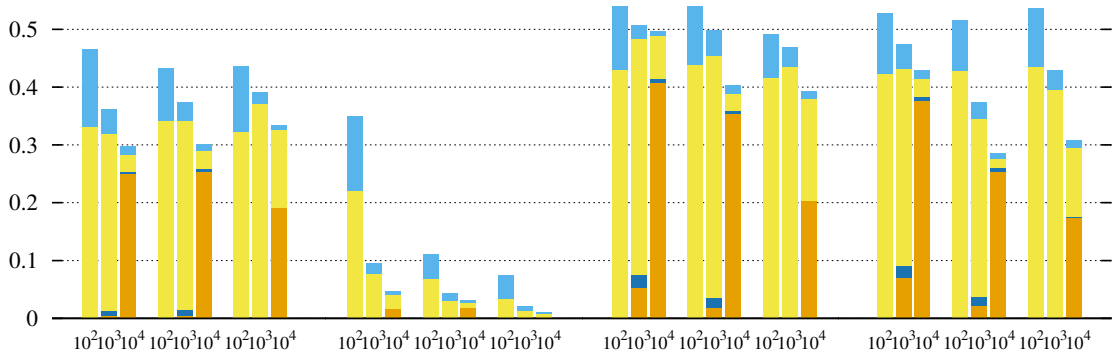
(e) Problem 5. (f) Problem 6. (g) Problem 7. (h) Problem 8.



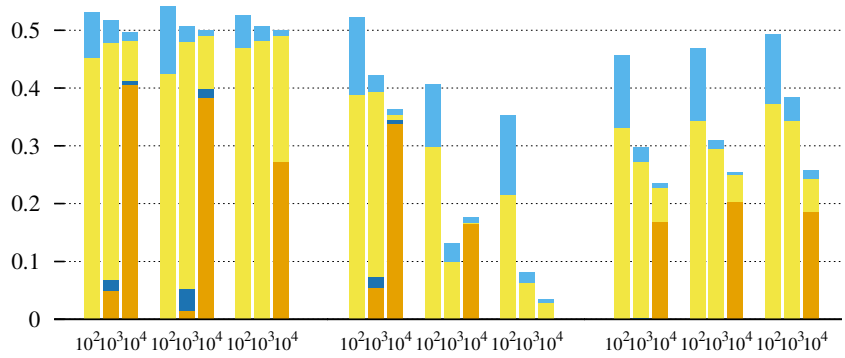
(i) Problem 9. (j) Problem 10. (k) Problem 11. (l) Problem 12.



(m) Problem 13. (n) Problem 14. (o) Problem 15. (p) Problem 16.



(q) Problem 17. (r) Problem 18. (s) Problem 19. (t) Problem 20.



(u) Problem 21. (v) Problem 22. (w) Problem 23.

Figure 2.2 – The training and testing errors (in resp. orange and yellow), as well as the standard deviations (in resp. dark blue and light blue) on each of the 23 image classification problems making up the Synthetic Visual Reasoning Test (SVRT). There are 3 groups of 3 histograms for each of the problem. Each group corresponds to the respective group of features, while the 3 histograms in a group corresponds to different amount of training examples (10^2, 10^3, 10^4). No histogram for 10^1 examples as the errors are all 0 in training and 0.5 in testing.

Table 2.1 – Mean training and testing errors with their standard deviations (all in percent) on all 23 problems of the Synthetic Visual Reasoning Test (SVRT), for the 3 feature groups and different numbers of training examples.

Number of	Group 1		Group 2		Group 3	
examples	Train	Test	Train	Test	Train	Test
10	0	49.1 ± 14.6	0	48.2 ± 14.8	0	49.6 ± 14.5
100	0	39.5 ± 4.98	0	37.2 ± 4.89	0	34.0 ± 4.56
1,000	1.95 ± 0.41	30.5 ± 1.50	0.55 ± 0.24	24.0 ± 1.50	0	23.0 ± 1.28
10,000	21.3 ± 1.49	25.8 ± 0.42	15.9 ± 2.31	19.1 ± 0.40	8.39 ± 1.22	15.8 ± 0.29

We also performed the same experiments using a Support Vector Machine (SVM) (Vapnik, 1995) with a Gaussian kernel, concatenating all the features together. We do not report the results as they show the same trends that the ones of figure 2.2, and are on average weaker (probably because the features were not weighted optimally).

The machine learning techniques we used for this study do not interpret the images as a configuration of parts, each with its own variability, and with a complex model of their relative positioning. That is why comparison with human subjects is embarrassing for the machine learning classifier, as most participants could solve all problems looking at a few pairs of examples only.

2.3 Conclusion

Both the number of training examples and the amount of features are critical to obtain good performances. Increasing the number of features is the most direct strategy to reduce the gap between humans and machine learning algorithms. The training time increasing at least linearly with both, it is crucial to develop smarter training algorithms, able to train a classifier efficiently in large scale scenarios.

3 Adaptive Sampling for Large Scale Boosting

In this chapter we present our contributions to reduce the training time of Boosting algorithms. Classical algorithms, such as AdaBoost, build a strong classifier without concern for the computational cost. Some applications, in particular in computer vision, may involve millions of training examples and very large feature spaces. In such contexts, the training time of off-the-shelf Boosting algorithms may become prohibitive. Several methods exist to accelerate training, typically either by sampling the features or the examples used to train the weak learners. Even if some of these methods provide a guaranteed speed improvement, they offer no insurance of being more efficient than any other, given the same amount of time.

*Our contributions are twofold: (a) a strategy to better deal with the increasingly common case where features come from multiple sources (*e.g. *color, shape, texture,* etc.*, in the case of images) and therefore can be partitioned into meaningful subsets; (b) new algorithms which estimate at every Boosting iteration the optimal trade-off between the number of weak learners and the number of training examples to look at in order to maximize the expected loss reduction. Experiments in image classification and object recognition on four standard computer vision datasets show that the adaptive methods we propose outperform basic sampling and state-of-the-art bandit methods. Content presented in this chapter is based on the following publications (Dubout and Fleuret, 2011a,b):*

> C. Dubout and F. Fleuret. Tasting families of features for image classification. In *International Conference on Computer Vision*, 2011.
>
> C. Dubout and F. Fleuret. Boosting with maximum adaptive sampling. In *Neural Information Processing Systems*, 2011.

3.1 Introduction

Boosting is a simple and efficient machine learning algorithm which provides state-of-the-art performance on many tasks. It consists of building a strong classifier as a linear combination of weak learners, by adding them one after another in a greedy manner.

It has been repeatedly demonstrated that combining multiple kind of features addressing different aspects of the signal is an extremely efficient strategy to improve performance (Opelt et al., 2006; Gehler and Nowozin, 2009; Dubout and Fleuret, 2011a,b). As shown by our experimental results, vanilla Boosting of stumps over multiple image features such as HOG, LBP, color histograms, *etc.*, usually reaches close to state-of-the-art performance. However, such techniques entails a considerable computational cost, which increases with the number of features considered during training.

The critical operations contributing to the computational cost of a Boosting iteration are the computations of the features and the selection of the weak learner. Both depend on the number of features and the number of training examples taken into account. While textbook AdaBoost repeatedly selects each weak learner using all the features and all the training examples for a predetermined number of rounds, one is not obligated to do so and can instead choose to look only at a subset of both.

Since performance increases with both, one needs to balance the two to keep the computational cost under control. As Boosting progresses, the performance of the candidate weak learners degrades, and they start to behave more and more similarly. While a small number of training examples is initially sufficient to characterize the good ones, as the learning problems become more and more difficult, optimal values for a fixed computational cost tend to move towards smaller number of features and larger number of examples.

In this paper, we present three new families of algorithms to explicitly address these issues: (1) Tasting (see § 3.4) uses a small number of features sampled prior to learning to adaptively bias the sampling towards promising subsets at every step; (2) Maximum Adaptive Sampling (see § 3.5.3) models the distribution of the weak learners' performance and the noise in order to determine the optimal trade-off between the number of weak learners and the number of examples to look at; and (3) Laminating (see § 3.5.4) iteratively refines the learner selection using more and more examples.

3.2 Related works

AdaBoost and similar Boosting algorithms estimate for each candidate weak learner a score dubbed "edge", which requires to loop through every training example and take into account its weight, which reflects its current importance in the loss reduction. Reducing this computational cost is crucial to cope with high-dimensional feature spaces or very large training sets. This can be achieved through two main strategies: sampling the training examples, or the

feature space, since there is a direct relation between features and weak learners.

Sampling the training set was introduced historically to deal with weak learners which cannot be trained with weighted examples (Freund and Schapire, 1996). This procedure consists of sampling examples from the training set according to their Boosting weights, and of approximating a weighted average over the full set by a non-weighted average over the sampled subset. It is related to Bootstrapping as similarly the training algorithm will sample harder and harder examples based on the performance of the previous weak learners. See § 3.3 for formal details. Such a procedure has been re-introduced recently for computational reasons (Bradley and Schapire, 2007; Duffield et al., 2007; Kalal et al., 2008; Fleuret and Geman, 2008), since the number of sampled examples controls the trade-off between statistical accuracy and computational cost.

Sampling the feature space is the central idea behind LazyBoost (Escudero et al., 2000), and simply consists of replacing the brute-force exhaustive search over the full feature set by an optimization over a subset produced by sampling uniformly a predefined number of features. The natural redundancy of most type of features makes such a procedure generally efficient. However, if a subset of important features is too small, it may be overlooked during training.

Recently developed algorithms rely on multi-arms bandit methods to balance properly the exploitation of features known to be informative, and the exploration of new features (Busa-Fekete and Kegl, 2009, 2010). The idea behind those methods is to associate a bandit arm to every feature, and to see the loss reduction as a reward. Maximizing the overall reduction is achieved with a standard bandit strategy such as UCB (Auer et al., 2002), or Exp3.P (Auer et al., 2003).

These techniques suffer from two important drawbacks. First they make the assumption that the quality of a feature – the expected loss reduction of a weak learner using it – is stationary. This goes against the underpinning of Boosting, which is that at any iteration the performance of the weak learners is relative to the Boosting weights, which evolve over the training (Exp3.P does not make such an assumption explicitly, but still rely exclusively on the history of past rewards). Second, without additional knowledge about the feature space, the only structure they can exploit is the stationarity of individual features. Hence, improvement over random selection can only be achieved by sampling again *the exact same features* already seen in the past. In our experiments, we therefore only use those methods in a context where features can be partitioned into subsets of different types. This allows us to model the quality, and thus to bias the sampling, at a higher level than individual features.

All those approaches exploit information about features to bias the sampling, hence making it more efficient, and reducing the number of weak learners required to achieve the same loss reduction. However, they do not explicitly aim at controlling the computational cost. In particular, there is no notion of varying the number of examples used for the estimation of the loss reduction.

3.3 Preliminaries

We first present in this section some analytical results to approximate a standard round of AdaBoost – or other similar Boosting algorithms – by sampling both the training examples and the features used to build the weak learners. We then precise more formally what we mean by subset of features or weak learners.

3.3.1 Standard Boosting

Given a binary training set

$$(\mathbf{x}_n, y_n) \in \mathcal{X} \times \{-1, 1\},\ n = 1, \ldots, N \tag{3.1}$$

where $\mathcal{X}$ is the space of the "visible" signal, and a set $\mathcal{H}$ of weak learners of the form $h : \mathcal{X} \to \{-1, 1\}$, the standard Boosting procedure consists of building a strong classifier

$$f = \sum_{t=1}^{T} \alpha_t h_t \tag{3.2}$$

by choosing the terms $\alpha_t \in \mathbb{R}$ and $h_t \in \mathcal{H}$ in a greedy manner so as to minimize a loss (*e.g.* the empirical exponential loss in the case of AdaBoost) estimated over the training examples. At every iteration, choosing the optimal weak learner boils down to finding the one with the largest edge ϵ, which is the derivative of the loss reduction w.r.t. the weak learner weight α. The higher this value, the more the loss can be reduced locally, and thus the better the weak learner. The edge is a linear function of the responses of the weak learner over the training examples

$$\epsilon(h) = \sum_{n=1}^{N} \omega(n) y_n h(\mathbf{x}_n) \tag{3.3}$$

where the weights $\omega(n)$s depend on the loss function (usually either the exponential or logistic loss) and on the current responses of f over the $\mathbf{x}_n$s. We consider without loss of generality that they have been normalized such that $\sum_{n=1}^{N} \omega(n) = 1$. We can therefore consider the weights $\omega(n)$s as a distribution over the training examples and rewrite the edge as an expectation

$$\epsilon(h) = \mathrm{E}_{N \sim \omega(n)} \left[y_N h(\mathbf{x}_N) \right] \tag{3.4}$$

where $N \sim \omega(n)$ stands for $\mathrm{P}(N = n) = \omega(n)$. The idea of weighting-by-sampling (Fleuret and Geman, 2008) consists of replacing the expectation in (3.4) with an approximation obtained by sampling. Let $N_1, \ldots, N_S$, be i.i.d. random variables distributed according to the discrete probability density distribution defined by the $\omega(n)$s, we define the approximated edge as

$$\hat{\epsilon}(h) = \frac{1}{S} \sum_{s=1}^{S} y_{N_s} h(\mathbf{x}_{N_s}) \tag{3.5}$$

which follows a binomial distribution centered on the true edge, with a variance decreasing with the number of sampled examples S. It is accurately modeled by the Gaussian

$$\hat{\epsilon}(h) \approx \mathcal{N}\left(\epsilon(h), \frac{(1+\epsilon(h))(1-\epsilon(h))}{S}\right) \tag{3.6}$$

as the approximation holds asymptotically and the magnitude of the weak learners' edges is typically small, such that $(1+\epsilon(h))(1-\epsilon(h)) \approx 1$.

3.3.2 Feature subsets

It frequently happens that the features making up the signal space $\mathcal{X}$ can be divided into meaningful disjoint subsets $\mathcal{F}_k$ such that $\mathcal{X} = \cup_{k=1}^{K} \mathcal{F}_k$. This division can for example be the result of the features coming from different sources or some natural clustering of the feature space. In such a case it makes sense to use this information during training, as features coming from the same subset $\mathcal{F}_k$ can typically be expected to be more homogeneous than features coming from different subsets.

3.4 Tasting

We describe here our approach called Tasting (Dubout and Fleuret, 2011a) which biases the sampling toward promising subsets of features. Tasting in its current form is limited to deal with weak learner looking at only one feature, such as decision stumps. Extending it to deal efficiently with weak learners looking at multiple features is outside of the scope of this work.

3.4.1 Main algorithm

The core idea of Tasting is to sample a small number R of features from every subset before starting the training *per se* and, at every Boosting step, in using these few features together with the current Boosting weights to get an estimate of the best subset(s) $\mathcal{F}_k$(s) to use.

We cannot stress enough that these R features are not the ones used to build the classifier, they are only used to figure out what is/are the best subset(s) at any time during training. As those sampled features are independent and identically distributed samples of the feature response vectors, we can compute the empirical mean of any functional of the said response vectors, in particular the expected loss reduction.

At any Boosting step, Tasting require, for any feature subset, an estimate of the expectation of the edge of the best weak learner we would obtain by sampling uniformly Q features from this subset and picking the best weak learner using one of them,

$$\mathrm{E}_{F_1,\dots,F_Q \sim \mathcal{U}(\mathcal{F}_k)} \left[\max_{q=1}^{Q} \max_{h \in \mathcal{H}_{F_q}} \epsilon(h) \right] \tag{3.7}$$

Algorithm 3.1 The Tasting 1.Q algorithm first samples uniformly R features from every subset $\mathscr{F}_k$. It uses these features at every Boosting step to find the optimal feature subset k^* from which to sample. After the selection of the Q features, the algorithm continues like AdaBoost.

Input: $\mathscr{F}$, Q, R, T

Initialize: $\forall k \in \{1,\dots,K\}$, $\forall r \in \{1,\dots,R\}$, $f_r^k \leftarrow \text{sample}(\mathscr{U}(\mathscr{F}_k))$

for $t = 1,\dots,T$ **do**

$\forall k \in \{1,\dots,K\}$, $\forall r \in \{1,\dots,R\}$, $\epsilon_r^k \leftarrow \max_{h \in \mathscr{H}_{f_r^k}} \epsilon(h)$

$k^* \leftarrow \operatorname{argmax}_k \mathrm{E}\left[\max_{q=1}^{Q} \epsilon_{R_q}^k\right]$ `# Computed using equation (3.10)`

$\forall q \in \{1,\dots,Q\}$, $F_q \leftarrow \text{sample}(\mathscr{U}(\mathscr{F}_{k^*}))$

$h_t \leftarrow \operatorname{argmax}_{h \in \cup_q \mathscr{H}_{F_q}} \epsilon(h)$

$\dots$

end for

where $\mathscr{F}_k$ are the indices of the features belonging to the k-th subset and $\mathscr{H}_F$ is the space of weak learners looking solely at feature F. Hence $\max_{h \in \mathscr{H}_{F_q}} \epsilon(h)$ is the best weak learner looking solely at feature F_q, and $\max_{q=1}^{Q} \max_{h \in \mathscr{H}_{F_q}} \epsilon(h)$ is the best weak learner looking solely at one of the Q features $F_1,\dots,F_Q$.

We can build an approximation of this quantity using the R features we have stored. Let $\epsilon_1,\dots,\epsilon_R$ be the edges of the best R weak learners built from these features. We make the assumption without loss of generality that $\epsilon_1 \le \epsilon_2 \le \dots \le \epsilon_R$. Let $R_1,\dots,R_Q$ be independent and identically distributed, uniform over $\{1,\dots,R\}$. We approximate the quantity (3.7) with

$$\mathrm{E}\left[\max_{q=1}^{Q} \epsilon_{R_q}\right] = \sum_{r=1}^{R} \mathrm{P}\left(\max_{q=1}^{Q} R_q = r\right) \epsilon_r \tag{3.8}$$

$$= \sum_{r=1}^{R} \left[\mathrm{P}\left(\max_{q=1}^{Q} R_q \le r\right) - \mathrm{P}\left(\max_{q=1}^{Q} R_q \le r-1\right)\right] \epsilon_r \tag{3.9}$$

$$= \frac{1}{R^Q} \sum_{r=1}^{R} \left[r^Q - (r-1)^Q\right] \epsilon_r. \tag{3.10}$$

3.4.2 Tasting variants

We propose two versions of the Tasting procedure, which differ in the number of feature subsets they visit at every iteration. Either one for Tasting 1.Q or up to Q for Tasting Q.1.

In Tasting 1.Q (algorithm 3.1), the selection of the optimal subset k^* from which to sample the Q features is accomplished by estimating for every subset the expected maximum edge, which is directly related to the expected loss reduction, if we were sampling from that subset only. The computation is done over the R features saved before starting training, which serve as a representation of the full set $\mathscr{F}_k$.

Algorithm 3.2 The Tasting Q.1 algorithm similarly starts by sampling uniformly R features from every subset $\mathscr{F}_k$, but it then uses them to find the optimal subset k_q^* for every one of the Q features to sample at every Boosting step. After the selection of the Q features, the algorithm continues like AdaBoost.

Input: $\mathscr{F}$, Q, R, T

Initialize: $\forall k \in \{1,\dots,K\}, \forall r \in \{1,\dots,R\}, f_r^k \leftarrow \text{sample}(\mathscr{U}(\mathscr{F}_k))$

for $t = 1,\dots,T$ **do**

$\forall k \in \{1,\dots,K\}, \forall r \in \{1,\dots,R\}, \epsilon_r^k \leftarrow \max_{h \in \mathscr{H}_{f_r^k}} \epsilon(h)$

$\epsilon^* \leftarrow 0$

for $q = 1,\dots,Q$ **do**

$k_q^* \leftarrow \operatorname{argmax}_k \mathrm{E}\left[\max\left(\epsilon^*, \epsilon^k\right)\right]$

$F_q \leftarrow \text{sample}\left(\mathscr{U}\left(\mathscr{F}_{k_q^*}\right)\right)$

$\epsilon^* \leftarrow \max\left(\epsilon^*, \max_{h \in \mathscr{H}_{F_q}} \epsilon(h)\right)$

end for

$h_t \leftarrow \operatorname{argmax}_{h \in \cup_q \mathscr{H}_{F_q}} \epsilon(h)$

...

end for

In Tasting Q.1 (see algorithm 3.2), it is not one but several feature subsets which can be selected, as the algorithm picks the best subset k_q^* for every one of the Q features to sample, given the best edge ϵ^* achieved so far. Again the computation is done only over the R features saved before starting training.

3.4.3 Relation with Bandit methods

The main strength of Boosting is its ability to spot and combine complementary features. If the loss has already been reduced in a certain "functional direction", the scores of weak learners in the same direction will be low, and they will be rejected. For instance, the firsts learners for a face detector may use color-based features to exploit the skin color. After a few Boosting steps using this modality, color would be would be exhausted as a source of information, and only examples with a non-standard face color would have large weights. Other features, for instance edge-based, would become more informative, and be picked.

Uniform sampling of features accounts poorly for such behavior since it simply discards the Boosting weights, and hence has no information whatsoever about the directions which have "already been exploited" and which should be avoided. In practice, this means that the rejection of bad feature can only be done at the level of the Boosting itself, which may end up with a majority of useless features.

Bandit methods (described in § 3.6.4) are slightly more adequate, as they model the performance of every feature from previous iterations. However, this modeling takes into account the Boosting weights very indirectly, as they make the assumption that the distributions of loss reduction are stationary, while they are precisely not. Coming back to our face-detector example, bandit methods would go on believing that color is informative since it was in the previous iterations, even if the Boosting weights have specifically accumulated on faces where color is now totally useless. While the estimate of loss reduction may asymptotically converge to an adequate model, it is a severe weakness while the Boosting weights are still evolving.

Tasting addresses this weakness by keeping the ability to properly estimate the performance of every feature subset, *given the current Boosting weights*, hence the ability to discard feature subsets redundant with features already picked. In some sense, Tasting can be seen as Boosting done at a the subset level.

3.5 Maximum Adaptive Sampling and Laminating

The algorithms in this section sample both the weak learners and the training examples at every iteration in order to maximize the expectation of the loss reduction, under a strict computational cost constraint.

3.5.1 Edge estimation

At every iteration they model the expectation of the edge of the selected weak learner. Let $\epsilon_1, \ldots, \epsilon_Q$ stand for the true edges of Q independently sampled weak learners. Let $\Delta_1, \ldots, \Delta_Q$ be a series of independent random variables standing for the noise in the estimation of the edges due to the sampling of only S training examples. Finally $\forall q$, let $\hat{\epsilon}_q = \epsilon_q + \Delta_q$ be the approximated edge. With these definitions, $\operatorname{argmax}_q \hat{\epsilon}_q$ is the selected weak learner. We define ϵ^* as the true edge of the selected weak learner, that is the one with the highest approximated edge

$$\epsilon^* = \epsilon_{\operatorname{argmax}_q \hat{\epsilon}_q}. \tag{3.11}$$

This quantity is random due to both the sampling of the weak learners, and the sampling of the training examples. The quantity we want to optimize is $\mathrm{E}[\epsilon^*]$, the expectation of the true edge of the selected learner, which increases with both Q and S. A higher Q increases the number of terms in the maximization of (3.11), while a higher S reduces the variance of the Δs, ensuring that ϵ^* is closer to $\max_q \epsilon_q$. In practice, if the variance of the Δs is of the order of, or higher than, the variance of the ϵs, the maximization is close to a random selection, and

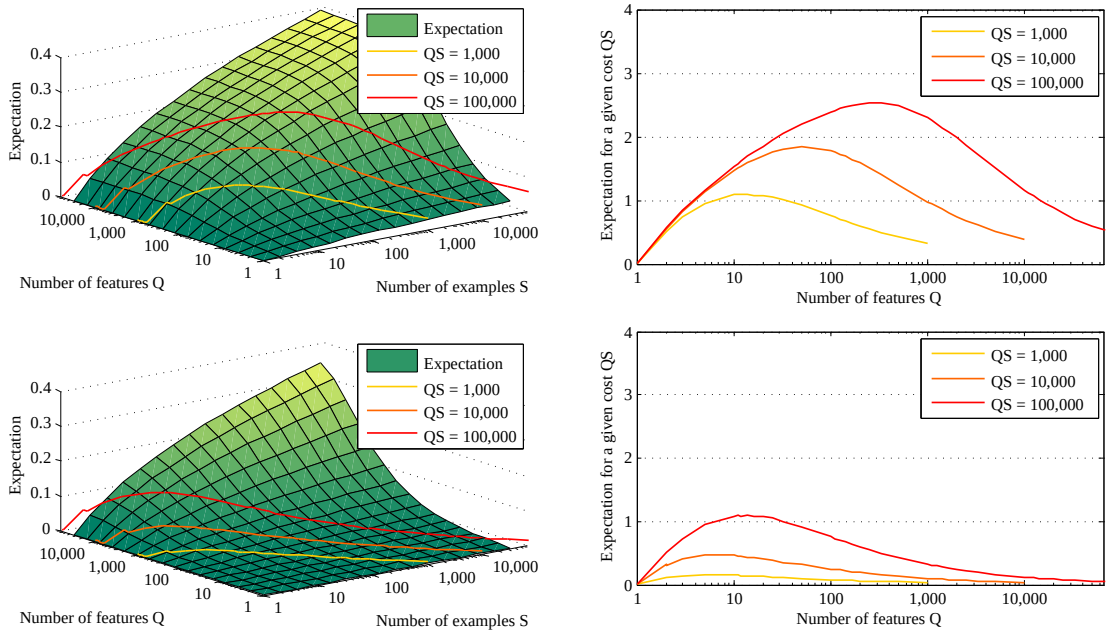


Figure 3.1 – Simulation of the expectation of ϵ^* in the case where both the ϵ_qs and the Δ_qs follow Gaussian distributions. Top: $\epsilon_q \sim \mathcal{N}(0, 10^{-2})$. Bottom: $\epsilon_q \sim \mathcal{N}(0, 10^{-4})$. In both simulations $\Delta_q \sim \mathcal{N}(0, \frac{1}{S})$. Left: expectation of ϵ^* vs. the number of sampled learners Q and the number of examples S. Right: same value as a function of Q alone, for different fixed costs (product of Q and S). As these graphs illustrate, the optimal value for Q is greater for larger variances of the ϵ_qs. In such a case the ϵ_qs are more spread out, and identifying the largest one can be done despite a large noise in the estimations, hence with a limited number of training examples.

looking at many weak learners is useless. Assuming that the $\hat{\epsilon}_q$s are all different we have

$$\mathrm{E}[\epsilon^*] = \mathrm{E}\left[\epsilon_{\mathrm{argmax}_q \hat{\epsilon}_q}\right] \tag{3.12}$$

$$= \sum_{q=1}^{Q} \mathrm{E}\left[\epsilon_q \prod_{i \neq q} \mathbf{1}_{\{\hat{\epsilon}_i < \hat{\epsilon}_q\}}\right] \tag{3.13}$$

$$= \sum_{q=1}^{Q} \mathrm{E}\left[\mathrm{E}\left[\epsilon_q \prod_{i \neq q} \mathbf{1}_{\{\hat{\epsilon}_i < \hat{\epsilon}_q\}} \,\middle|\, \hat{\epsilon}_q\right]\right] \tag{3.14}$$

$$= \sum_{q=1}^{Q} \mathrm{E}\left[\mathrm{E}\left[\epsilon_q \,\middle|\, \hat{\epsilon}_q\right] \prod_{i \neq q} \mathrm{E}\left[\mathbf{1}_{\{\hat{\epsilon}_i < \hat{\epsilon}_q\}} \,\middle|\, \hat{\epsilon}_q\right]\right] \tag{3.15}$$

where the last equality follows from the independence of the weak learners.

3.5.2 Modeling the true edge

If the distributions of the ϵ_qs and the Δ_qs are Gaussians or mixtures of Gaussians, we can derive analytical expressions for both $\mathrm{E}[\epsilon_q \,|\, \hat{\epsilon}_q]$ and $\mathrm{E}\left[\mathbf{1}_{\{\hat{\epsilon}_i < \hat{\epsilon}_q\}} \,\middle|\, \hat{\epsilon}_q\right]$, and compute the value of

$\mathrm{E}[\epsilon^*]$ efficiently. In the case where weak learners can be partitioned into meaningful subsets, it makes sense to model the distributions of the edges separately for each subset.

As an illustrative example, we consider here the case where the ϵ_qs, the Δ_qs, and hence also the $\hat{\epsilon}_q$s all follow Gaussian distributions. We take $\epsilon_q \sim \mathcal{N}(0,1)$ and $\Delta_q \sim \mathcal{N}(0,\sigma^2)$ and obtain

$$\mathrm{E}[\epsilon^*] = Q\,\mathrm{E}\left[\mathrm{E}[\epsilon_1 \,|\, \hat{\epsilon}_1] \prod_{i\neq 1} \mathrm{E}\left[\mathbf{1}_{\{\hat{\epsilon}_i < \hat{\epsilon}_1\}} \,\middle|\, \hat{\epsilon}_1\right]\right] \tag{3.16}$$

$$= Q\,\mathrm{E}\left[\frac{\hat{\epsilon}_1}{\sigma^2+1}\Phi\left(\frac{\hat{\epsilon}_1}{\sqrt{\sigma^2+1}}\right)^{Q-1}\right] \tag{3.17}$$

$$= \frac{Q}{\sqrt{\sigma^2+1}}\,\mathrm{E}\left[\epsilon_1 \Phi(\epsilon_1)^{Q-1}\right] \tag{3.18}$$

$$= \frac{1}{\sqrt{\sigma^2+1}}\,\mathrm{E}\left[\max_{1\leq q\leq Q} \epsilon_q\right] \tag{3.19}$$

where Φ stands for the cumulative distribution function of the unit Gaussian, and σ typically depends on S. See figure 3.1 for an illustration of the behavior of $\mathrm{E}[\epsilon^*]$ for two different variances of the ϵ_qs and a cost proportional to QS, the total number of features computed.

There is no reason to expect the distribution of the ϵ_qs to be Gaussian, contrary to the Δ_qs, as shown in (3.6), but this is not a problem as it can always be approximated by a mixture, for which we can still derive analytical expressions, even if the ϵ_qs or the Δ_qs have different distributions for different qs.

3.5.3 M.A.S. variants

We created three algorithms modeling the distribution of the ϵ_qs with a Gaussian mixture model, and $\Delta_q = \hat{\epsilon}_q - \epsilon_q$ as a Gaussian. The first one, M.A.S. naive, is described in Algorithm 3.3, and fits the model to the edges estimated at the previous iteration.

The second one, M.A.S. 1.Q, takes into account the decomposition of the weak learners into K subsets, associated to different kind of features. It models the distributions of the ϵ_qs separately for each subset, estimating the distribution of each on a small number of weak learners and examples sampled at the beginning of each Boosting iteration, chosen so as to account for 10% of the total computational cost. From these models, it optimizes Q, S, and the index k of the subset to sample from. Unlike M.A.S. naive, it has to draw a small number of weak learners and examples in order to fit the model since the edges estimated at the previous iterations came from a unique subset.

Finally M.A.S. Q.1 similarly models the distributions of the ϵ_qs, but it optimizes $Q_1, \ldots, Q_K$ greedily, starting from $Q_1 = 0, \ldots, Q_K = 0$, and iteratively incrementing one of the Q_k so as to maximize $\mathrm{E}[\epsilon^*]$. This greedy procedure is repeated for different values of S and ultimately the $Q_1, \ldots, Q_K, S$ leading to the maximum expectation are selected.

Algorithm 3.3 The M.A.S. naive algorithm models the current edge distribution with a Gaussian mixture model fitted on the edges estimated at the previous iteration. It uses this density model to compute the pair (Q^*, S^*) maximizing the expectation of the true edge of the selected learner $\mathrm{E}[\epsilon^*]$, and then samples the corresponding number of weak learners and training examples, before keeping the weak learner with the highest approximated edge. After the selection of the Q features, the algorithm continues like AdaBoost.

Input: $gmm, Cost$

for $t = 1, \ldots, T$ **do**

$(Q^*, S^*) \leftarrow \underset{\mathrm{cost}(Q,S) \leq Cost}{\mathrm{argmax}} \mathrm{E}_{gmm}[\epsilon^*]$

$\forall q \in \{1, \ldots, Q^*\}, H_q \leftarrow \mathrm{sample}(\mathcal{U}(\mathcal{H}))$

$\forall s \in \{1, \ldots, S^*\}, N_s \leftarrow \mathrm{sample}(\mathcal{U}(\{1, \ldots, N\}))$

$\forall q \in \{1, \ldots, Q^*\}, \hat{\epsilon}_q \leftarrow \frac{1}{S^*} \sum_{s=1}^{S^*} y_{N_s} H_q(x_{N_s})$ `# Similar to equation (3.5)`

$h_t \leftarrow H_{\mathrm{argmax}_q \hat{\epsilon}_q}$

$gmm \leftarrow \mathrm{fit}(\hat{\epsilon}_1, \ldots, \hat{\epsilon}_{Q^*})$

...

end for

3.5.4 Laminating

The last algorithm we have developed tries to reduce the requirement for a density model of the ϵ_qs. At every Boosting iteration it iteratively reduces the number of considered weak learners, and increases the number of examples taken into account.

Given fixed Q and S, at every Boosting iteration, the Laminating algorithm first samples Q weak learners and S training examples. Then, it computes the approximated edges and keeps the $\frac{Q}{2}$ best learners. If more than one remains, it samples $2S$ examples, and re-iterates. The whole process is described in Algorithm 3.4. The number of iterations is bounded by $\lceil \log_2(Q) \rceil$.

We have the following results on the accuracy of this Laminating procedure (the proof is given in Appendix A):

Lemma 1 *Let $q^\star = \mathrm{argmax}_q \epsilon_q$ and $\delta > 0$. The probability for an iteration of the Laminating algorithm to retain only weak learners with edges below or equal to $\epsilon_{q^\star} - \delta$ is*

$$\mathrm{P}\left(\left|\left\{q : \epsilon_q \leq \epsilon_{q^\star} - \delta, \hat{\epsilon}_q \geq \max_{r : \epsilon_r \geq \epsilon_{q^\star} - \delta} \hat{\epsilon}_r\right\}\right| \geq \frac{Q}{2}\right) \leq 4 \exp\left(-\frac{\delta^2 S}{2}\right).$$

This holds regardless of the independence of the ϵ_qs and/or the Δ_qs.

Since at each iteration the number of examples S doubles the lemma implies the following theorem (the proof is given in Appendix B):

Theorem 1 *The probability for the full Laminating procedure starting with Q weak learners*

Algorithm 3.4 The Laminating algorithm starts by sampling Q weak learners and S examples at the beginning of every Boosting iteration, and refine those by successively halving the number of learners and doubling the number of examples until only one learner remains. After the selection of the Q features, the algorithm continues like AdaBoost.

Input: Q, S
  for $t = 1, \ldots, T$ **do**
    $\forall q \in \{1, \ldots, Q\}, H_q \leftarrow \text{sample}(\mathcal{U}(\mathcal{H}))$
    while $Q > 1$ **do**
      $\forall s \in \{1, \ldots, S\}, N_s \leftarrow \text{sample}(\mathcal{U}(\{1, \ldots, N\}))$
      $\forall q \in \{1, \ldots, Q\}, \hat{\epsilon}_q \leftarrow \frac{1}{S} \sum_{s=1}^{S} y_{N_s} H_q(x_{N_s})$ `# Similar to equation (3.5)`
      $\text{sort}(H_1, \ldots, H_Q)$ s.t. $\hat{\epsilon}_1 \geq \cdots \geq \hat{\epsilon}_Q$ `# Order the weak learners s.t.`
      $Q \leftarrow \frac{Q}{2}$ `# the best half comes first`
      $S \leftarrow 2S$
    end while
    ...
  end for

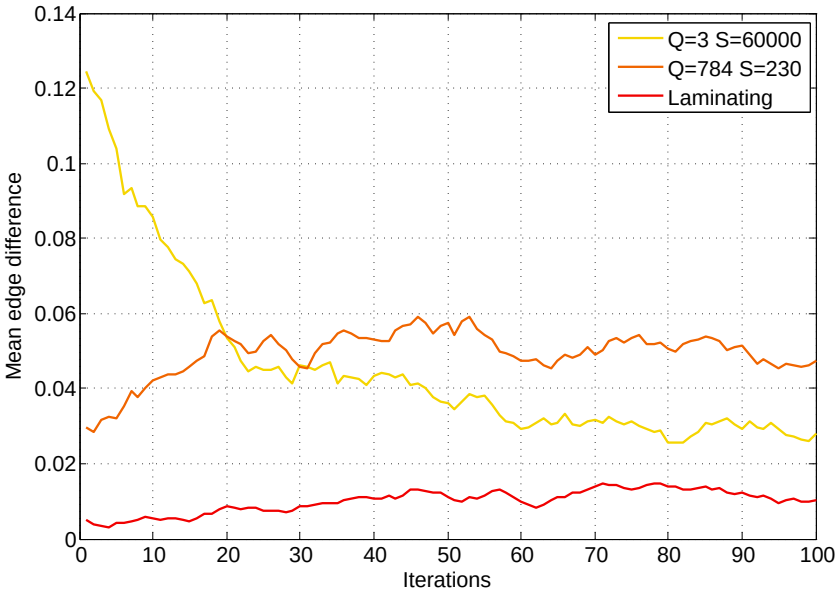
and S examples to end up with a learner with an edge below or equal to $\epsilon_{q^*} - \delta$ (the edge of the optimal weak learner at the start of the procedure minus δ) is upper bounded by (the proof is given in Appendix B)[1]

$$\frac{4}{1 - \exp\left(-\frac{\delta^2 S}{69}\right)} - 4.$$

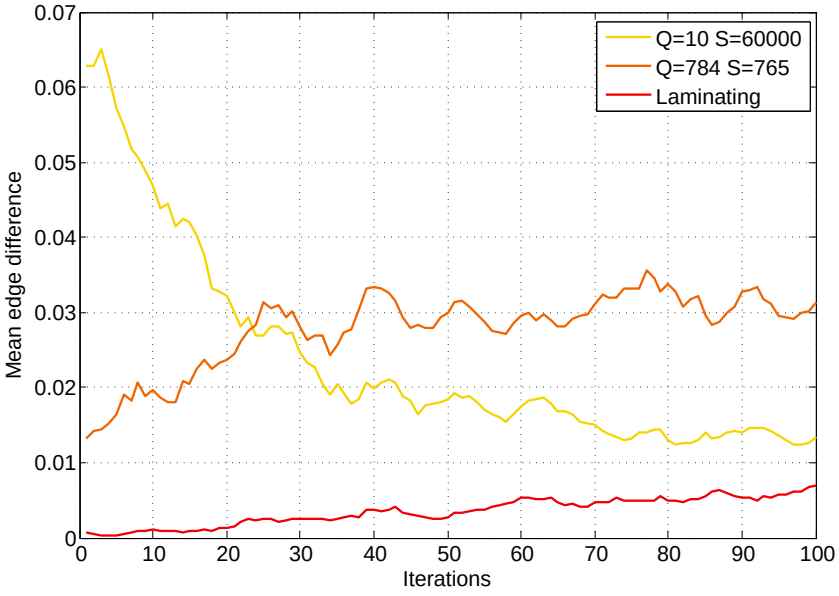
The theorem shows that as the number of samples grows, the probability to retain a bad weak learner eventually goes down exponentially with the number of training examples, as in this case the bound can be approximated by $4 \exp\left(-\frac{\delta^2 S}{69}\right)$. This confirms the usual relation between the number of examples and the complexity of the space of predictors in learning theory.

In practice the difference between the maximum edge $\epsilon_{q^\star}$ and the edge of the final weak learner selected by Laminating is typically smaller than the difference with the edge of a learner selected by a strategy looking at a fixed number of weak learners and training examples, as can be observed in figure 3.2.

[1] We thank Gilles Blanchard for suggesting an improvement of our initial result to get a bound which does not depend on Q.



(a) $QS = 180,000$.



(b) $QS = 600,000$.

Figure 3.2 – Difference between the maximum edge and the best edge found by 3 different sampling strategies on the MNIST dataset using the original features. The algorithm used is AdaBoost.MH using $T = 100$ decision stumps as weak learners, and the results were averaged over 10 randomized runs. The first strategy samples uniformly a small number of features Q and determines the best one using all $S = 60,000$ training examples. The second strategy samples all $Q = 784$ features and determines the best one using a small number of training examples S. The third strategy is Laminating, starting from all the features and a suitable number of training examples chosen so as to have the same cost as the first two strategies. The cost is the product of Q and S and is set to $QS = 180,000$ in (1) and $QS = 600,000$ in (b).

3.6 Experiments

We demonstrate the effectiveness of our approaches on four standard image classification and object detection datasets, using 19 kinds of features (33 on Caltech 101) divided in as many subsets. We used the AdaBoost.MH algorithm (Schapire and Singer, 1999) with decision stumps as weak learners to be able to use all methods in the same conditions.

3.6.1 Features

The features used in our experiments with all but the Caltech 101 dataset can be divided into three categories. (1) Image transforms: identity, grayscale conversion, Fourier and Haar transforms, gradient image, local binary patterns (ILBP/LBP). (2) Intensity histograms: sums of the intensities in random image patches, grayscale and color histograms of the entire image. (3) Gradient histograms: histograms of (oriented and non oriented) gradients, Haar-like features.

The features from the first category typically have a large dimensionality, usually proportional to the number of pixels in the image. Some of them do not pre-process the images (identity, grayscale conversion, LBP, *etc.*) while some pre-transform them to another space, prior to accessing any feature (typically the Fourier and Haar transforms).

Features from the second and third categories being histograms, they are usually much smaller (containing typically of the order of a few hundreds to a few thousands coefficients), but require some pre-processing to build the histograms.

For the Caltech 101 dataset we used the same features as (Gehler and Nowozin, 2009) in their experiments. They used five type of features: PHOG shape descriptors, appearance (SIFT) descriptors, region covariance, local binary patterns, and V1S+, which are normalized Gabor filters. More details can be found in the referenced paper. Those features are computed in a spatial pyramid, where each scale of the pyramid is considered as being part of a different subset, leading to a total of 33 features. The number of features used in our experiments (33) differ from (Gehler and Nowozin, 2009) as they also compute a 'subwindow-kernel' of SIFT features which we did not use.

3.6.2 Datasets

The first dataset that we used is the MNIST handwritten digits database (LeCun et al., 1998b). It is composed of 10 classes and its training and testing sets consist respectively of 60,000 and 10,000 grayscale images of resolution 28×28 pixels (see the upper left part of figure 3.3 for some examples). The total number of features on this dataset is 16,775.

The second dataset that we used is the INRIA Person dataset (Dalal and Triggs, 2005). It is composed of a training and a testing set respectively of 2,418 and 1,126 color images of

Figure 3.3 – Example images from the four datasets used for the experimental validation. Top left: first image of every digit taken from the MNIST database. Top right: images from the INRIA Person dataset. Bottom left: random images from the Caltech 101 dataset. Bottom right: some of the first images of the CIFAR-10 dataset.

pedestrians of dimensions 64×128 pixels cropped from real-world photographs, along with 1,219 and 453 "background" images not containing any people (see the upper right part of figure 3.3 for some examples). We extracted 10 negative samples from each one of the background image, following the setup of (Dalal and Triggs, 2005). The total number of features on this dataset is 230,503.

The third dataset that we used is Caltech 101 (Fei-Fei et al., 2004) due to its wide usage and the availability of already computed features (Gehler and Nowozin, 2009). It consists of color images of various dimensions organized in 101 object classes (see the bottom left part of figure 3.3 for some examples). We sampled 15 training examples and 20 distinct test examples from every class, as advised on the dataset website. The total number of features on this dataset is 360,630.

The fourth and last dataset that we used is CIFAR-10 (Krizhevsky, 2009). It is a labeled subset of the 80 tiny million images dataset. It is composed of 10 classes and its training and testing sets consist respectively of 50,000 and 10,000 color images of size 32×32 pixels (see the bottom left part of figure 3.3 for some examples). The total number of features on this dataset is 29,879.

3.6.3 Uniform sampling baselines

A naive sampling strategy would pick the Q features uniformly in $\cup_k \mathcal{F}_k$. However, this does not distribute the sampling properly among the $\mathcal{F}_k$s. In the extreme case, if one of the $\mathcal{F}_k$ had a far greater cardinality than the others, all features would come from it. And in most contexts, mixing features from the different $\mathcal{F}_k$s in an equilibrate manner is critical to benefit from their complementarity. We propose the four following baselines to pick a good feature at

every Boosting step:

- **Best subset** picks Q features at random in a fixed subset, the one with the smallest final Boosting loss.
- **Uniform Naive** picks Q features at random, uniformly in $\cup_k \mathscr{F}_q$.
- **Uniform 1.Q** picks one of the feature subsets at random, and then samples the Q features from that single subset.
- **Uniform Q.1** picks at random, uniformly, Q subsets of features (with replacement if $Q > K$), and then picks one feature uniformly in each subset.

The cost of running **Best subset** is K times higher than running the other three strategies since the subset leading to the smallest final Boosting loss is not known a priori. Also, since it makes use of one subset only we can expect its final performance to be lower than the others. It was included for comparison only.

3.6.4 Bandit sampling baselines

The strategies of the previous section are purely random and do not exploit any kind of information to bias their sampling. Smarter strategies to deal with the problem of exploration-exploitation trade-off were first introduced in (Busa-Fekete and Kegl, 2009), and extended in (Busa-Fekete and Kegl, 2010). The driving idea of these papers is to entrust a multi-armed bandits (MAB) algorithm (respectively UCB in (Auer et al., 2002) and Exp3.P in (Auer et al., 2003)) with the mission to sample useful features.

The multi-armed bandits problem is defined as follows: there are M gambling machines (i.e. the "arms" of the bandits), and at every time-step t the gambler chooses an arm j_t, pulls it, and receives a reward $r_{j_t}^t \in [0,1]$. The goal of the algorithm is to minimize the weak-regret, that is the difference between the reward obtained by the gambler and the best fixed arm, retrospectively.

The first weakness of these algorithms in the context of accelerating Boosting, identified in § 3.2, is the assumption of stationarity of the loss reduction, which cannot be easily dealt with. Even though the Exp3.P algorithm does not make such an assumption explicitly, it still ignores the Boosting weights, and thus can only rely on the history of past rewards.

The second weakness, the application context, can be addressed in our setting by learning the usefulness of the subsets instead of individual features.

A third weakness is that in Boosting one aims at minimizing the loss (which translates into maximizing the sum of the rewards for the bandit algorithm), and not at minimizing the weak-regret.

Finally, another issue arises when trying to use those algorithms in practice. As they use some kind of confidence intervals, the scale of the rewards matters greatly. For example, if all the rewards obtained are very small ($\forall t, r^t \leq \epsilon \ll 1$), the algorithms will not learn anything, as they expect rewards to make full use of the range $[0,1]$.

For this reason we set the bandit baselines' meta-parameters to the ones leading to the lowest loss a posteriori, as explained in § 3.6.5, and use a third multi-armed bandit algorithm in our experiments, ϵ-greedy (Auer et al., 2002), which does not suffer from this problem.

Hence, we use in our experiments the three following baselines, using the same reward as in (Busa-Fekete and Kegl, 2010):

- **UCB** picks Q features from the subset that maximizes $\bar{r}_j + \sqrt{(2\log n)/n_j}$, where $\bar{r}_j$ is the current average reward of subset j, n_j is the number of times subset j was chosen so far, and n is the current Boosting round.
- **Epx3.P** maintains a distribution of weights over the feature subsets, and at every round picks one subset accordingly, obtains a reward, and updates the distribution. For the precise definition of the algorithm, see (Auer et al., 2003; Busa-Fekete and Kegl, 2010).
- **ϵ-greedy** picks Q features from the subset with the highest current average reward with probability $1-\epsilon_n$, or from a random subset with probability ϵ_n, where $\epsilon_n = \frac{cK}{d^2 n}$, and c and d are parameters of the algorithm.

3.6.5 Results

We tested all the proposed methods of § 3.4, § 3.5.3, and § 3.5.4 against the baselines described in § 3.6.3 and § 3.6.4 on the four benchmark datasets described above in § 3.6.2 using the standard train/test cuts and all the features of § 3.6.1. We report the results of doing up to 10,000 Boosting rounds averaged through ten randomized runs in tables 3.1–3.8 and figures 3.4–3.11. We used as cost for all the algorithms the number of evaluated features, that is for each Boosting iteration QS, the number of sampled features times the number of sampled examples. For the Laminating algorithm we multiplied this cost by the number of iterations $\lceil \log_2(Q) \rceil$. We set the maximum cost of all the algorithms to $10N$, setting $Q = 10$ and $S = N$ for the baselines, as this configuration leads to the best results after 10,000 Boosting rounds.

The parameters of the baselines – namely the scale of the rewards for UCB and Exp3.P, and the c/d^2 ratio of ϵ-greedy – were optimized by trying all values of the form 2^n, $n = \{0, 1, ..., 11\}$, and keeping the one leading to the smallest final Boosting loss on the training set, which is unfair to the uniform baselines as well as our methods. We set the values of the parameters of Exp3.P to $\eta = 0.3$ and $\lambda = 0.15$ as recommended in (Busa-Fekete and Kegl, 2010).

Tasting requires only one parameter to be set, the number R of features to initially store from each family. We used the value $R = 100$ in all our experiments, but we observed that setting it

to 10 only marginally affects them, increasing the test error by less than 0.02% on average, and reducing the (logarithm of) the loss by less than 3%.

These results illustrate the efficiency of the proposed methods. Up to 1,000 Boosting rounds, the Laminating algorithms is the clear winner on three out of the four datasets. Then come the M.A.S. and the Tasting procedures, still performing far better than the baselines. On the Caltech 101 dataset the situation is different. Since it contains a much smaller number of training examples compared to the other datasets (1515 versus several tens of thousands), there is no advantage in sampling examples. It even proves detrimental as the M.A.S. and Laminating methods are beaten by the baselines after 1,000 iterations.

The performance of all the methods tends to get similar for 10,000 stumps, which is unusually large. The Tasting algorithm appears to fare the best, sampling examples offering no speed gain for such a large number of Boosting steps, except on the INRIA Person dataset. On this dataset the Laminating algorithm still dominates, although its advantage in loss reduction does not translate into a lower test error anymore.

3.7 Conclusion

We have improved Boosting by modeling the statistical behavior of the weak learners' edges. This allowed us to maximize the loss reduction under strict control of the computational cost. Experiments demonstrate that the algorithms perform well on real-world classification tasks.

Extensions of the proposed methods could be investigated along two axes. The first one is to merge the best two methods by adding a Tasting component to the Laminating procedure, in order to bias the sampling towards promising feature subsets. The second is to add a bandit-like component to the methods by adding a variance term related to the lack of samples, and their obsolescence in the Boosting process. This would account for the degrading density estimation when subsets have not been sampled for a while, and induce an exploratory sampling which may be missing in the current algorithms.

Table 3.1 – Mean Boosting loss ($\log_{10}$) after various number of steps on MNIST with all families of features (standard deviations are in between parentheses). Methods with a $\star$ require the tuning of meta-parameters, which have been optimized by training multiple times.

	$T = 10$	$T = 100$	$T = 1,000$	$T = 10,000$
Best family*	-0.426 (0.018)	-0.947 (0.014)	-1.84 (0.008)	-4.84 (0.006)
Uniform Naive	-0.379 (0.010)	-0.847 (0.015)	-1.74 (0.010)	-5.37 (0.014)
Uniform 1.Q	-0.355 (0.025)	-0.750 (0.015)	-1.51 (0.011)	-3.90 (0.020)
Uniform Q.1	-0.384 (0.021)	-0.857 (0.017)	-1.72 (0.006)	-5.06 (0.015)
UCB*	-0.398 (0.019)	-0.786 (0.009)	-1.64 (0.008)	-5.54 (0.009)
Exp3.P*	-0.363 (0.030)	-0.769 (0.034)	-1.66 (0.043)	-5.42 (0.032)
ϵ-greedy*	-0.371 (0.023)	-0.877 (0.020)	-1.78 (0.035)	-5.45 (0.100)
Tasting 1.Q	-0.427 (0.020)	-0.963 (0.012)	-1.91 (0.007)	**-5.90 (0.013)**
Tasting Q.1	-0.437 (0.017)	-0.968 (0.013)	-1.91 (0.009)	**-5.91 (0.020)**
M.A.S. Naive	-0.508 (0.019)	-1.008 (0.011)	-1.80 (0.015)	-5.06 (0.016)
M.A.S. 1.Q	-0.475 (0.021)	-0.978 (0.016)	-1.74 (0.008)	-4.15 (0.020)
M.A.S. Q.1	-0.429 (0.025)	-0.981 (0.009)	-1.78 (0.010)	-4.51 (0.018)
Laminating	**-0.549 (0.009)**	**-1.099 (0.009)**	**-2.00 (0.005)**	-5.87 (0.014)

Table 3.2 – Mean test error (%) after various number of Boosting steps on MNIST with all families of features (standard deviations are in between parentheses). Methods with a $\star$ require the tuning of meta-parameters, which have been optimized by training multiple times.

	$T = 10$	$T = 100$	$T = 1,000$	$T = 10,000$
Best family*	36.5 (3.56)	5.77 (0.24)	1.47 (0.073)	**0.916 (0.038)**
Uniform Naive	45.3 (2.12)	7.80 (0.50)	1.64 (0.079)	0.931 (0.050)
Uniform 1.Q	49.4 (5.01)	10.80 (0.83)	2.18 (0.096)	1.076 (0.059)
Uniform Q.1	43.0 (3.61)	7.40 (0.49)	1.70 (0.108)	0.970 (0.048)
UCB*	41.9 (4.46)	9.67 (0.32)	1.86 (0.077)	0.940 (0.048)
Exp3.P*	47.9 (5.98)	10.27 (1.24)	1.79 (0.124)	**0.923 (0.055)**
ϵ-greedy*	45.9 (5.24)	7.04 (0.62)	1.57 (0.145)	**0.882 (0.030)**
Tasting 1.Q	36.0 (2.66)	5.38 (0.23)	**1.41 (0.106)**	0.920 (0.040)
Tasting Q.1	34.7 (2.49)	5.31 (0.27)	**1.36 (0.068)**	0.938 (0.036)
M.A.S. Naive	26.3 (2.07)	4.78 (0.15)	1.54 (0.074)	0.960 (0.047)
M.A.S. 1.Q	29.9 (2.90)	5.21 (0.42)	1.63 (0.082)	1.036 (0.039)
M.A.S. Q.1	35.7 (3.50)	5.21 (0.19)	1.68 (0.060)	1.013 (0.052)
Laminating	**21.9 (0.85)**	**3.85 (0.21)**	**1.35 (0.077)**	0.964 (0.049)

Table 3.3 – Mean Boosting loss ($\log_{10}$) after various number of steps on INRIA Person with all families of features (standard deviations are in between parentheses). Methods with a $\star$ require the tuning of meta-parameters, which have been optimized by training multiple times.

	$T = 10$	$T = 100$	$T = 1,000$	$T = 10,000$
Best family*	-0.338 (0.037)	-0.935 (0.039)	-3.72 (0.038)	-26.9 (0.091)
Uniform Naive	-0.309 (0.022)	-0.861 (0.054)	-3.92 (0.074)	-31.9 (0.305)
Uniform 1.Q	-0.304 (0.031)	-1.009 (0.029)	-4.86 (0.131)	-40.0 (0.388)
Uniform Q.1	-0.304 (0.031)	-1.009 (0.029)	-4.86 (0.131)	-40.0 (0.388)
UCB*	-0.349 (0.029)	-1.081 (0.045)	-5.47 (0.142)	-49.3 (0.288)
Exp3.P*	-0.307 (0.027)	-0.915 (0.041)	-4.53 (0.106)	-44.7 (0.560)
ϵ-greedy*	-0.344 (0.015)	-1.113 (0.080)	-5.92 (0.180)	-49.3 (0.856)
Tasting 1.Q	-0.398 (0.026)	-1.297 (0.034)	-6.54 (0.117)	-55.1 (0.499)
Tasting Q.1	-0.398 (0.026)	-1.297 (0.034)	-6.54 (0.117)	-55.1 (0.499)
M.A.S. Naive	-0.459 (0.025)	-1.502 (0.039)	-7.23 (0.106)	-60.4 (0.415)
M.A.S. 1.Q	-0.413 (0.052)	-1.454 (0.063)	-6.87 (0.059)	-55.9 (0.322)
M.A.S. Q.1	-0.413 (0.052)	-1.454 (0.063)	-6.87 (0.059)	-55.9 (0.322)
Laminating	**-0.558 (0.034)**	**-2.054 (0.052)**	**-11.24 (0.109)**	**-99.9 (0.205)**

Table 3.4 – Mean test error (%) after various number of Boosting steps on INRIA Person with all families of features (standard deviations are in between parentheses). Methods with a $\star$ require the tuning of meta-parameters, which have been optimized by training multiple times.

	$T = 10$	$T = 100$	$T = 1,000$	$T = 10,000$
Best family*	12.18 (1.55)	3.29 (0.525)	1.195 (0.097)	1.004 (0.046)
Uniform Naive	13.40 (0.83)	4.87 (0.400)	1.268 (0.167)	0.532 (0.050)
Uniform 1.Q	13.96 (1.29)	3.92 (0.405)	0.691 (0.088)	0.331 (0.038)
Uniform Q.1	13.96 (1.29)	3.92 (0.405)	0.691 (0.088)	0.331 (0.038)
UCB*	12.05 (1.24)	3.17 (0.242)	0.613 (0.046)	**0.304 (0.040)**
Exp3.P*	13.61 (1.61)	4.09 (0.470)	0.794 (0.086)	0.324 (0.037)
ϵ-greedy*	12.88 (0.75)	2.89 (0.475)	0.537 (0.073)	0.343 (0.038)
Tasting 1.Q	11.24 (0.98)	2.33 (0.175)	0.566 (0.035)	0.320 (0.033)
Tasting Q.1	11.24 (0.98)	2.33 (0.175)	0.566 (0.035)	0.320 (0.033)
M.A.S. Naive	8.80 (0.82)	1.66 (0.118)	**0.438 (0.056)**	**0.269 (0.034)**
M.A.S. 1.Q	10.12 (1.26)	1.82 (0.227)	0.497 (0.053)	**0.276 (0.027)**
M.A.S. Q.1	10.12 (1.26)	1.82 (0.227)	0.497 (0.053)	**0.276 (0.027)**
Laminating	**6.85 (0.73)**	**1.12 (0.087)**	**0.389 (0.044)**	0.301 (0.026)

Table 3.5 – Mean Boosting loss ($\log_{10}$) after various number of steps on Caltech 101 with all families of features (standard deviations are in between parentheses). Methods with a $\star$ require the tuning of meta-parameters, which have been optimized by training multiple times.

	$T = 10$	$T = 100$	$T = 1,000$	$T = 10,000$
Best family*	-0.798 (0.003)	-1.44 (0.010)	-7.17 (0.042)	-65.5 (0.41)
Uniform Naive	-0.791 (0.005)	-1.40 (0.008)	-6.81 (0.036)	-61.8 (0.39)
Uniform 1.Q	-0.786 (0.010)	-1.36 (0.011)	-5.84 (0.065)	-49.6 (0.45)
Uniform Q.1	-0.809 (0.003)	-1.44 (0.008)	-6.74 (0.052)	-59.2 (0.51)
UCB*	-0.814 (0.002)	-1.40 (0.007)	-6.46 (0.061)	-61.6 (0.54)
Exp3.P*	-0.786 (0.012)	-1.34 (0.017)	-5.89 (0.072)	-54.4 (0.60)
ϵ-greedy*	-0.810 (0.005)	-1.42 (0.014)	-7.26 (0.068)	-67.1 (0.54)
Tasting 1.Q	**-0.823 (0.004)**	**-1.50 (0.009)**	**-7.47 (0.057)**	**-68.1 (0.44)**
Tasting Q.1	**-0.822 (0.004)**	**-1.50 (0.007)**	**-7.46 (0.024)**	**-68.1 (0.35)**
M.A.S. Naive	-0.803 (0.005)	-1.43 (0.009)	-6.70 (0.046)	-59.1 (0.51)
M.A.S. 1.Q	-0.779 (0.002)	-1.01 (0.006)	-2.04 (0.033)	-29.5 (0.42)
M.A.S. Q.1	-0.796 (0.003)	-1.21 (0.008)	-5.01 (0.123)	-42.7 (0.65)
Laminating	-0.813 (0.004)	-1.43 (0.011)	-6.33 (0.091)	-54.4 (0.55)

Table 3.6 – Mean test error (%) after various number of Boosting steps on Caltech 101 with all families of features (standard deviations are in between parentheses). Methods with a $\star$ require the tuning of meta-parameters, which have been optimized by training multiple times.

	$T = 10$	$T = 100$	$T = 1,000$	$T = 10,000$
Best family*	95.2 (0.87)	79.4 (1.36)	56.7 (1.05)	41.9 (0.60)
Uniform Naive	95.8 (0.41)	80.3 (1.27)	55.6 (0.83)	38.8 (0.85)
Uniform 1.Q	95.9 (0.78)	79.0 (0.97)	54.2 (0.75)	40.8 (0.93)
Uniform Q.1	**94.2 (0.53)**	76.5 (1.28)	51.8 (0.94)	37.6 (1.06)
UCB*	**94.2 (0.90)**	78.6 (0.70)	52.6 (1.21)	37.0 (1.11)
Exp3.P*	95.8 (0.87)	80.3 (0.82)	54.7 (0.75)	40.6 (0.94)
ϵ-greedy*	94.8 (0.48)	76.7 (1.07)	**50.6 (0.65)**	37.4 (0.96)
Tasting 1.Q	**93.8 (0.90)**	**74.2 (0.71)**	**50.7 (1.22)**	**35.3 (0.83)**
Tasting Q.1	**93.9 (0.82)**	**74.5 (1.34)**	**50.5 (0.52)**	**35.5 (0.99)**
M.A.S. Naive	**94.3 (0.65)**	76.2 (1.26)	51.8 (1.20)	37.9 (1.06)
M.A.S. 1.Q	96.4 (0.73)	90.5 (0.78)	85.9 (0.75)	53.6 (1.03)
M.A.S. Q.1	95.2 (0.93)	85.7 (0.24)	58.8 (0.61)	44.5 (0.98)
Laminating	**94.3 (0.63)**	77.0 (1.10)	53.0 (0.76)	38.4 (1.05)

Table 3.7 – Mean Boosting loss ($\log_{10}$) after various number of steps on CIFAR 10 with all families of features (standard deviations are in between parentheses). Methods with a $\star$ require the tuning of meta-parameters, which have been optimized by training multiple times.

	$T = 10$	$T = 100$	$T = 1,000$	$T = 10,000$
Best family*	-0.268 (0.005)	-0.331 (0.002)	-0.434 (0.001)	-0.666 (0.000)
Uniform Naive	-0.262 (0.005)	-0.337 (0.003)	-0.478 (0.001)	-0.928 (0.001)
Uniform 1.Q	-0.255 (0.006)	-0.331 (0.004)	-0.467 (0.003)	-0.843 (0.001)
Uniform Q.1	-0.266 (0.004)	-0.344 (0.002)	-0.486 (0.001)	-0.911 (0.001)
UCB*	-0.271 (0.004)	-0.336 (0.002)	-0.486 (0.002)	-0.899 (0.001)
Exp3.P*	-0.256 (0.007)	-0.329 (0.003)	-0.475 (0.002)	-0.860 (0.008)
ϵ-greedy*	-0.261 (0.005)	-0.349 (0.003)	-0.490 (0.003)	-0.884 (0.009)
Tasting 1.Q	-0.276 (0.002)	-0.360 (0.002)	-0.501 (0.001)	-0.946 (0.002)
Tasting Q.1	-0.277 (0.003)	-0.362 (0.002)	**-0.503 (0.001)**	**-0.950 (0.001)**
M.A.S. Naive	-0.278 (0.004)	-0.354 (0.003)	-0.489 (0.001)	-0.912 (0.001)
M.A.S. 1.Q	-0.280 (0.003)	-0.350 (0.003)	-0.447 (0.002)	-0.632 (0.002)
M.A.S. Q.1	-0.279 (0.002)	-0.351 (0.002)	-0.446 (0.002)	-0.619 (0.001)
Laminating	**-0.290 (0.002)**	**-0.373 (0.001)**	-0.498 (0.001)	-0.880 (0.001)

Table 3.8 – Mean test error (%) after various number of Boosting steps on CIFAR 10 with all families of features (standard deviations are in between parentheses). Methods with a $\star$ require the tuning of meta-parameters, which have been optimized by training multiple times.

	$T = 10$	$T = 100$	$T = 1,000$	$T = 10,000$
Best family*	73.6 (1.47)	57.4 (0.52)	44.8 (0.22)	40.2 (0.20)
Uniform Naive	74.9 (1.09)	55.9 (0.70)	38.9 (0.46)	32.2 (0.35)
Uniform 1.Q	76.6 (1.45)	57.5 (1.00)	39.9 (0.46)	31.3 (0.30)
Uniform Q.1	74.3 (1.62)	53.8 (0.23)	37.6 (0.42)	30.9 (0.27)
UCB*	73.3 (1.18)	56.2 (0.48)	37.7 (0.52)	30.6 (0.36)
Exp3.P*	77.2 (2.67)	58.0 (0.70)	38.9 (0.56)	**30.3 (0.32)**
ϵ-greedy*	75.8 (1.89)	53.4 (0.67)	37.1 (0.49)	**30.0 (0.34)**
Tasting 1.Q	72.6 (1.02)	50.9 (0.49)	**36.2 (0.29)**	31.7 (0.24)
Tasting Q.1	71.8 (1.00)	50.9 (0.70)	**36.3 (0.44)**	31.5 (0.20)
M.A.S. Naive	71.9 (1.08)	52.5 (0.38)	37.5 (0.23)	31.0 (0.31)
M.A.S. 1.Q	70.7 (1.42)	53.3 (0.84)	40.5 (0.44)	33.8 (0.34)
M.A.S. Q.1	71.4 (0.80)	52.7 (0.46)	40.4 (0.47)	34.1 (0.24)
Laminating	**67.8 (0.92)**	**49.1 (0.41)**	36.8 (0.20)	31.5 (0.32)

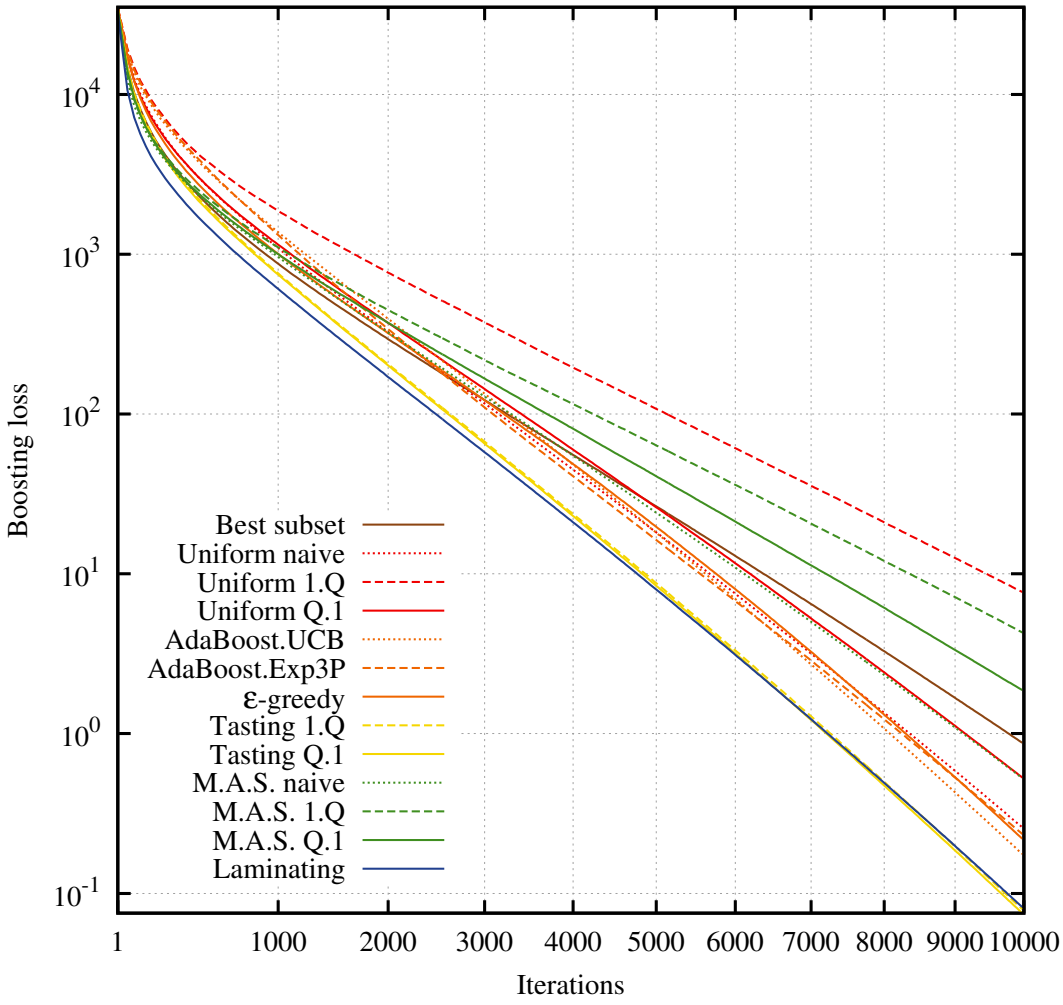


Figure 3.4 – Mean Boosting loss on the MNIST dataset.

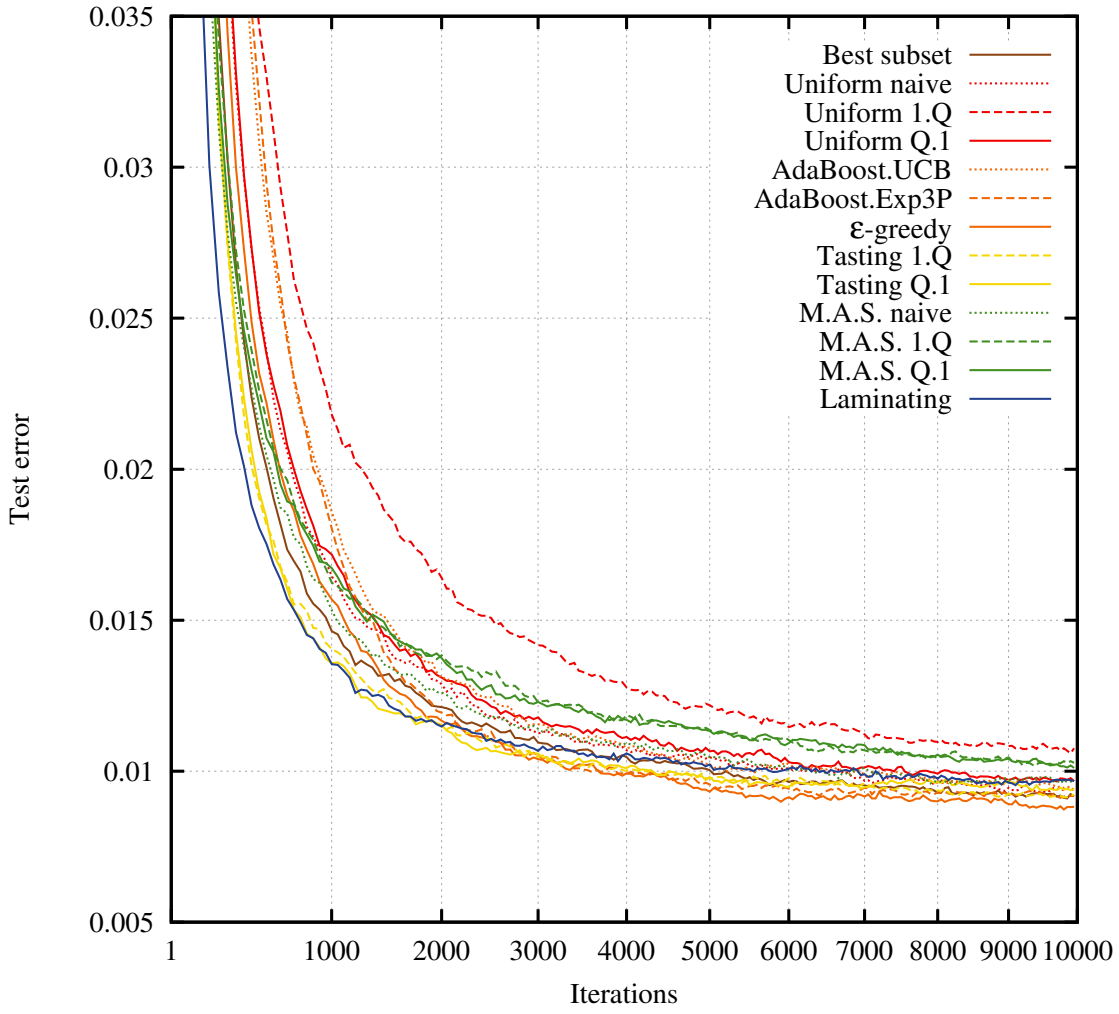


Figure 3.5 – Mean test error on the MNIST dataset.

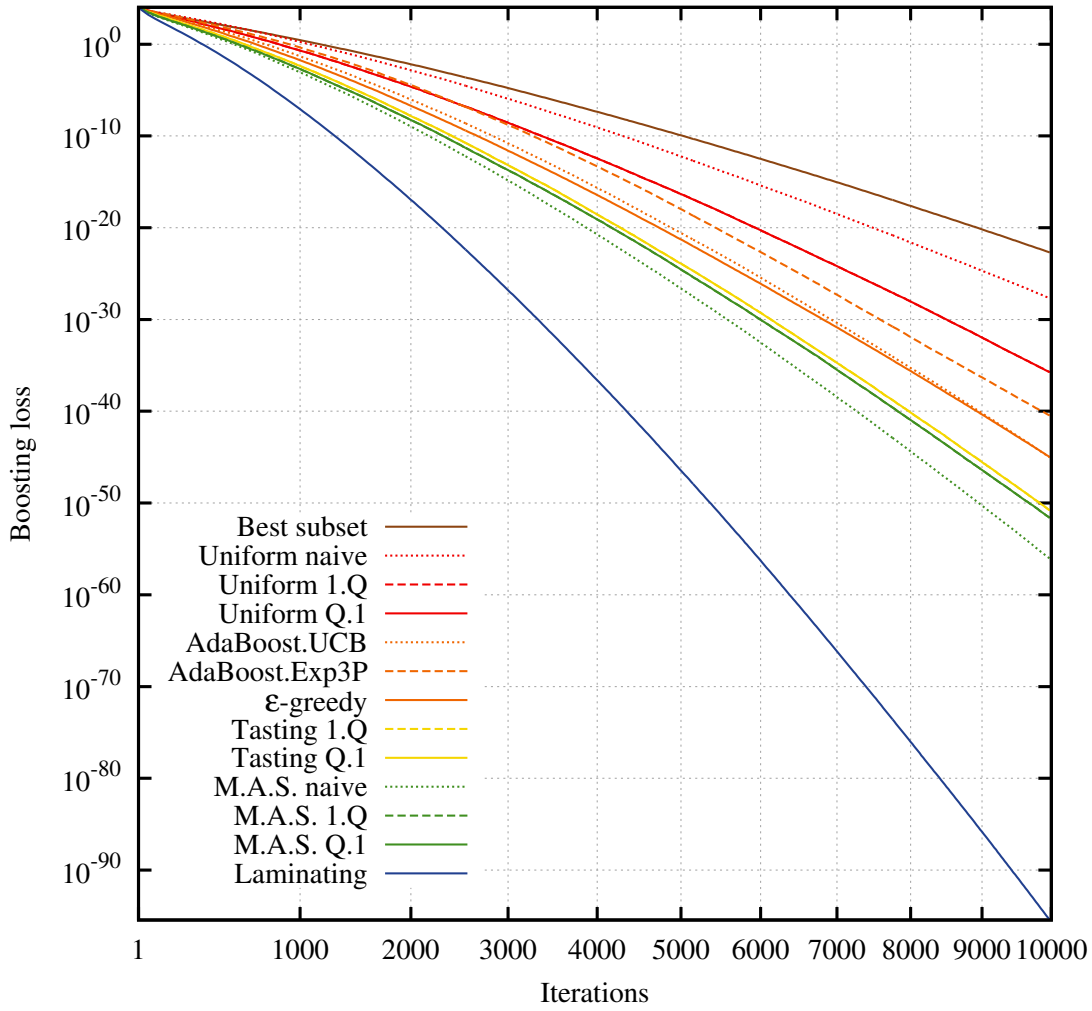


Figure 3.6 – Mean Boosting loss on the INRIA Person dataset.

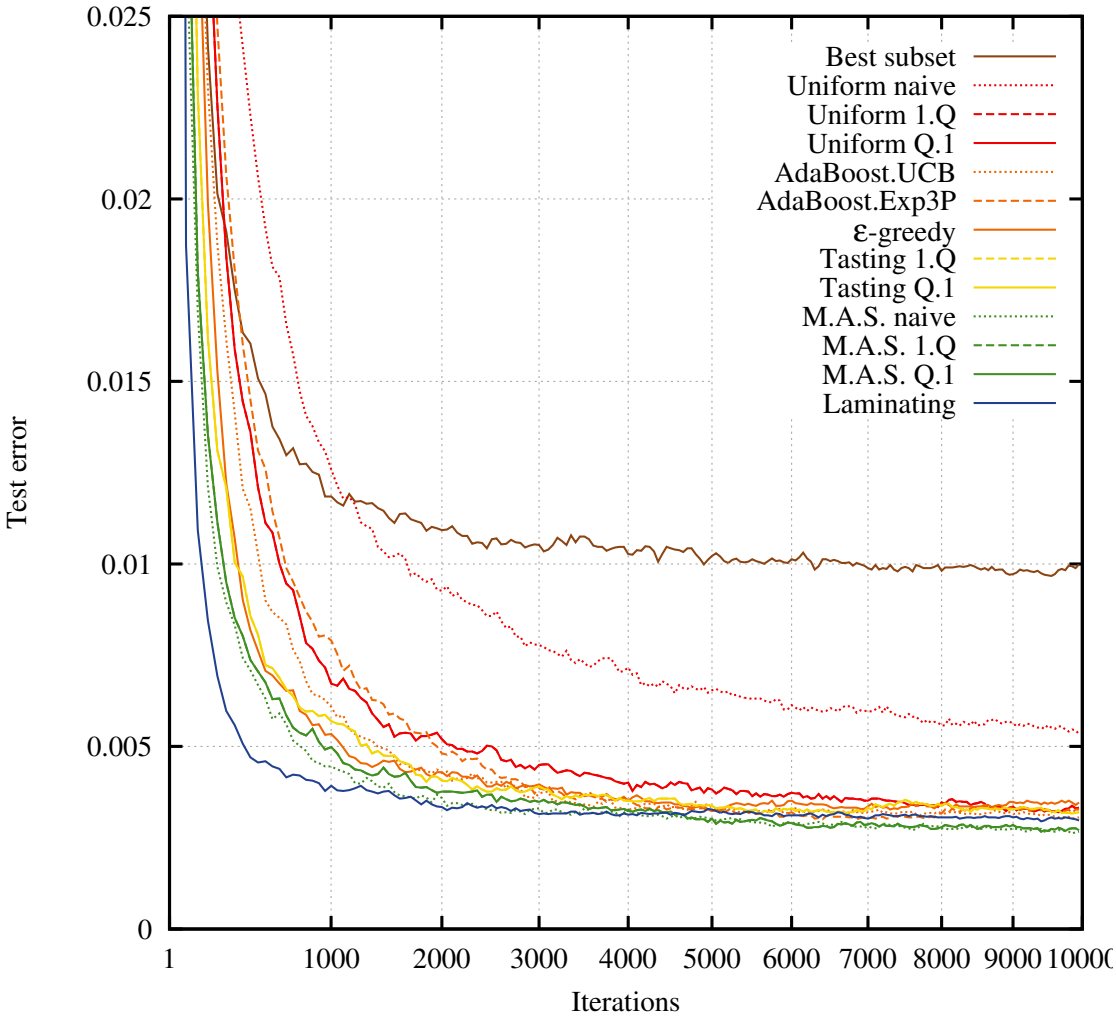


Figure 3.7 – Mean test error on the INRIA Person dataset.

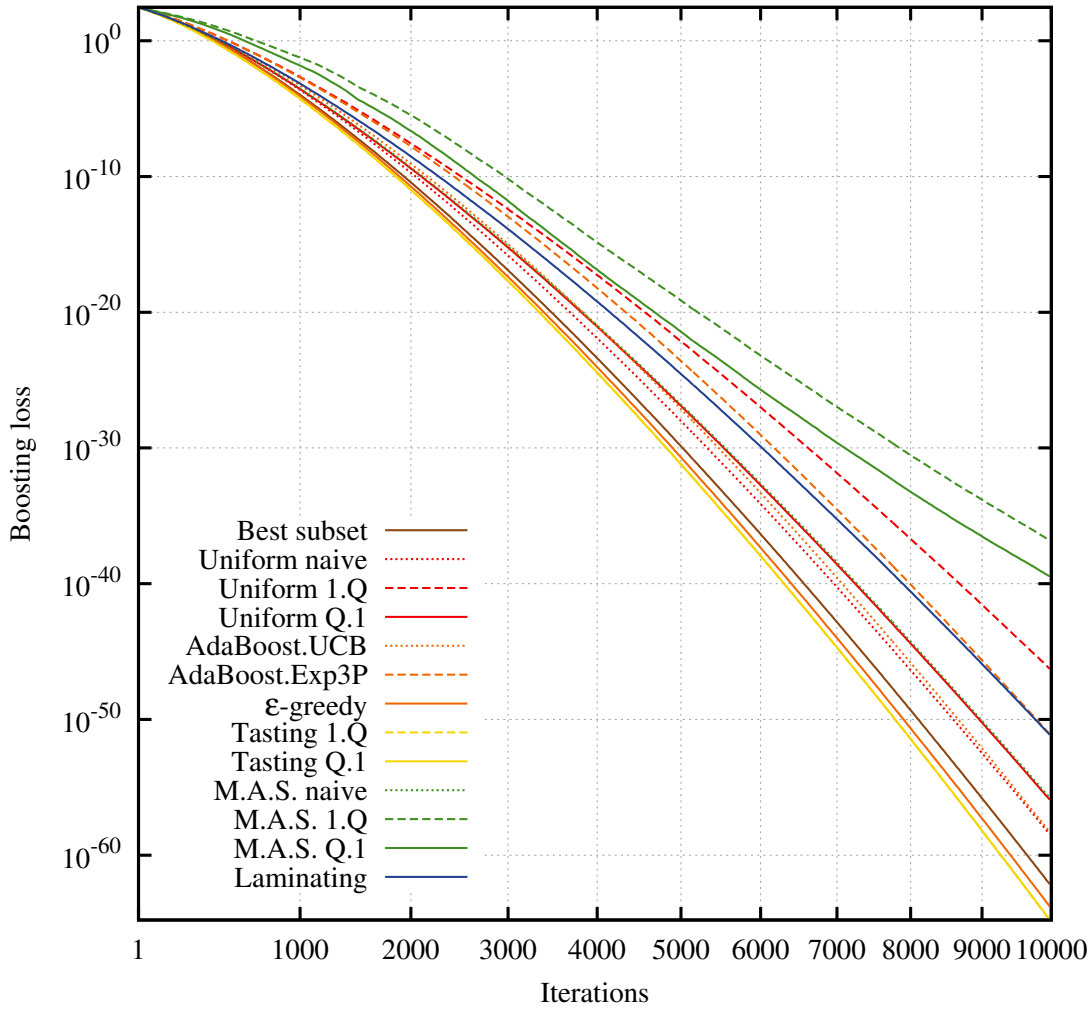


Figure 3.8 – Mean Boosting loss on the Caltech 101 dataset.

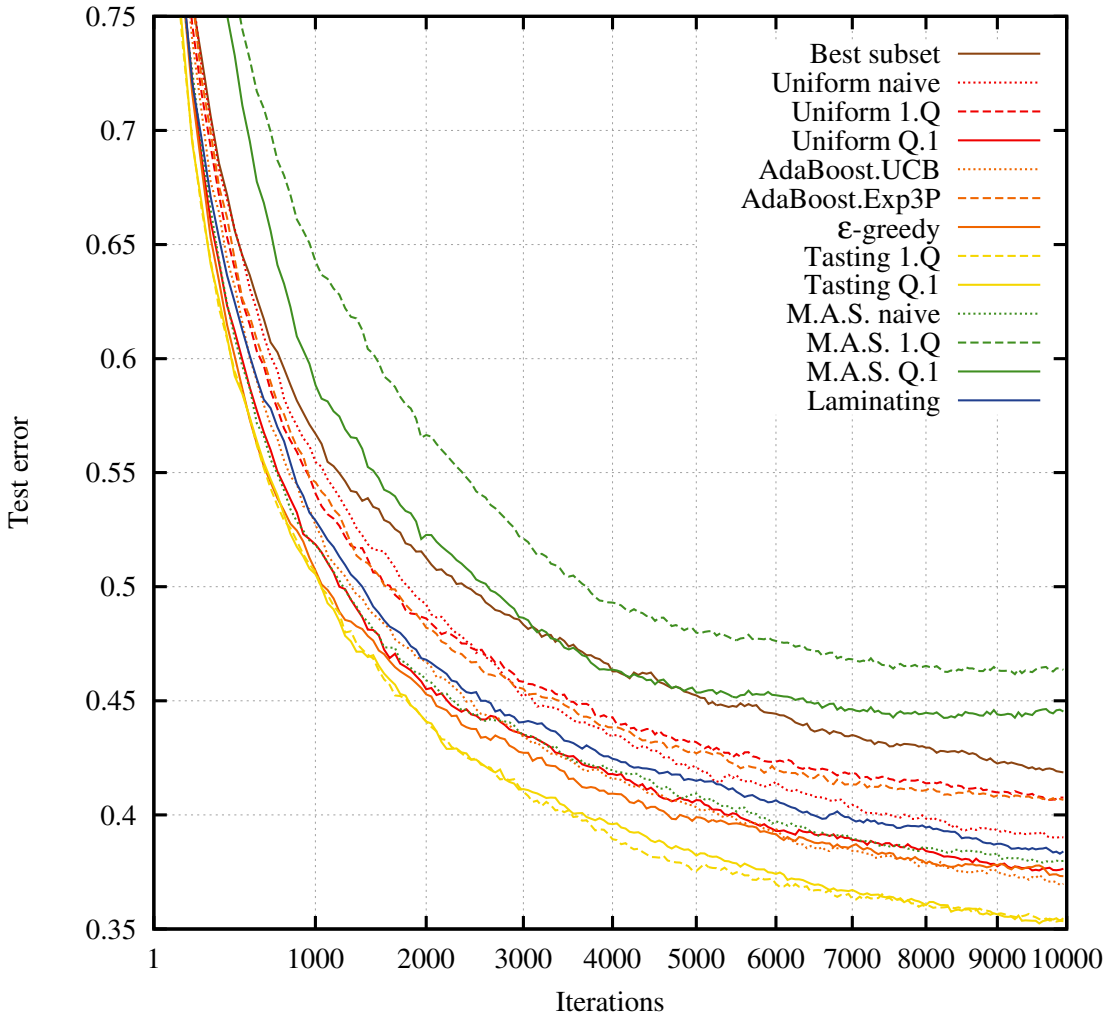


Figure 3.9 – Mean test error on the Caltech 101 dataset.

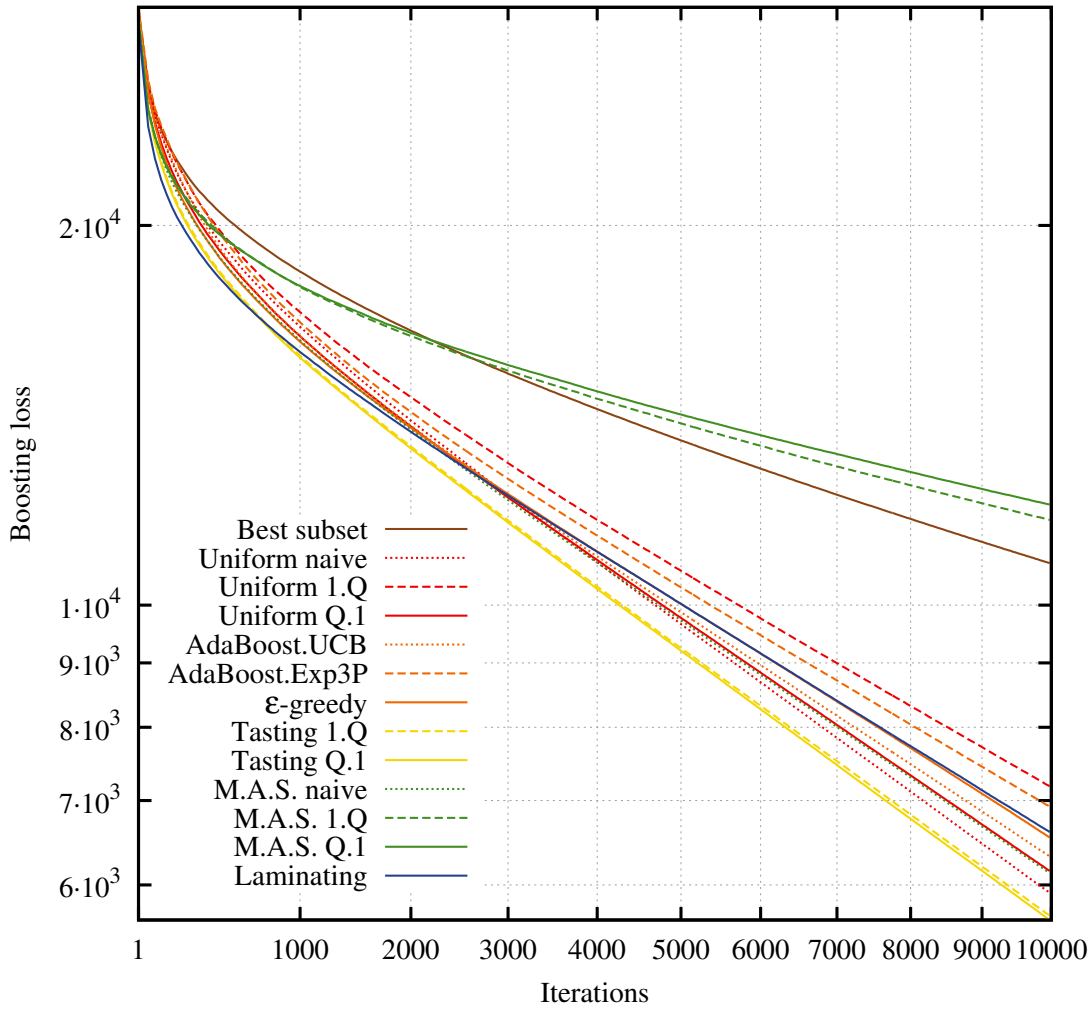


Figure 3.10 – Mean Boosting loss on the CIFAR-10 dataset.

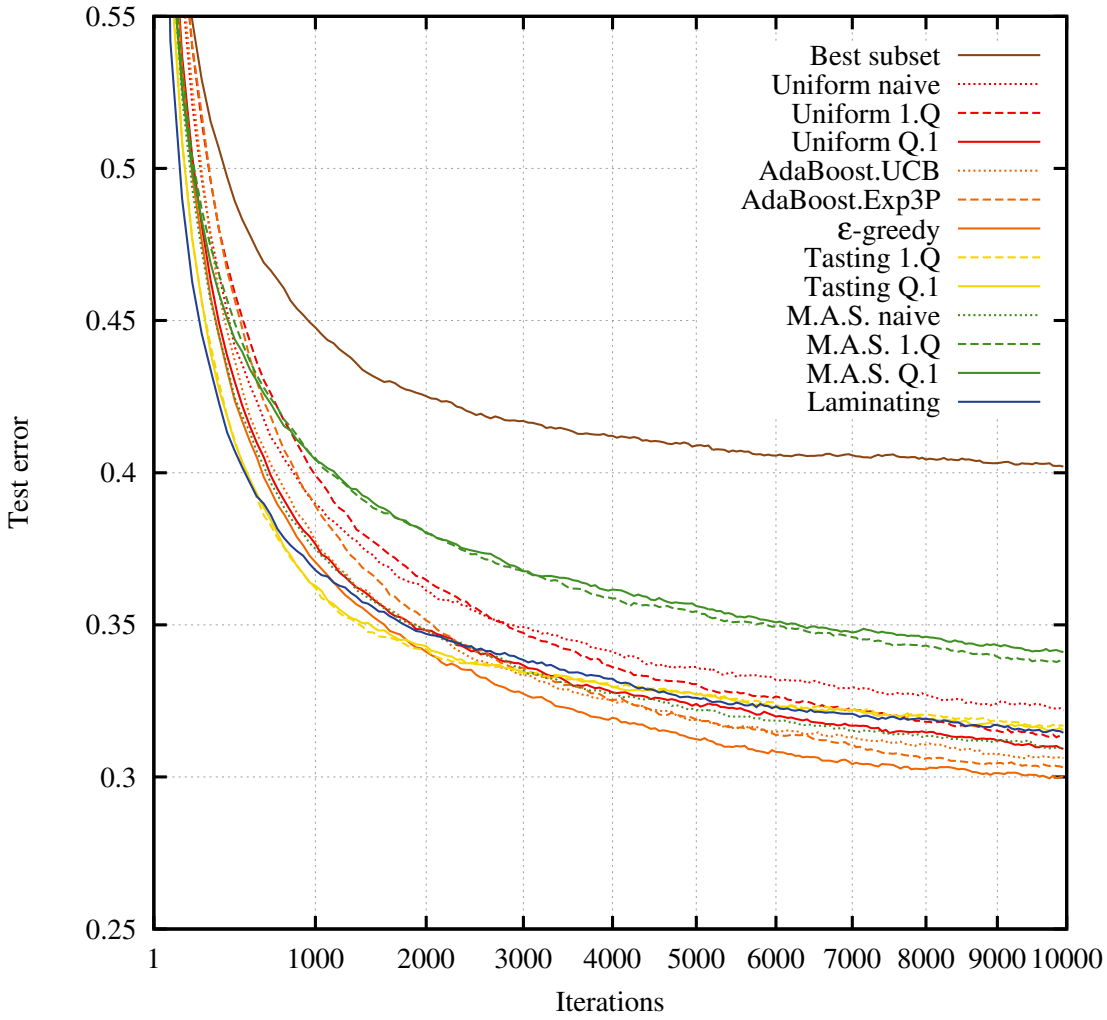


Figure 3.11 – Mean test error on the CIFAR-10 dataset.

Part II Object Detection in High Dimensional Feature Space

4 Accelerated Evaluation of Linear Object Detectors

In this chapter we present our contributions to speed up the evaluation of a broad range of linear object detectors. The main bottleneck of many of those systems is the computational cost of the convolutions between the features extracted from the image to process, and the linear filters. The intuition underpinning our strategy is to replace convolutions by point-wise multiplications in the Fourier domain, to reuse forward transforms across images and filters, and to exploit the linearity property of the Fourier transform to reduce the number of inverse transforms. This linearity property allows us to switch the order of summations and inverse Fourier transforms (which are expensive to compute as they are in $\Theta(N \log N)$ *and require to shuffle a lot of data around), reducing the number of inverse transforms drastically. Additional advantages of using the Fourier transform to compute convolutions besides computational efficiency are its better numerical accuracy and its cost remaining constant with respect to the filter size. Content presented in this chapter is based on the following publication (Dubout and Fleuret, 2012a):*

> C. Dubout and F. Fleuret. Exact Acceleration of Linear Object Detectors. In *Computer Vision – ECCV 2012*, volume 7574 of *Lecture Notes in Computer Science*, pages 301–311. Springer Berlin Heidelberg, 2012.

4.1 Introduction

A common technique for object detection is to apply a binary classifier at every possible position and scale of an image in a sliding-window fashion. However, searching the entire search space, even with a simple detector can be slow, especially if a large number of image features are used.

To that end, linear classifiers have gained a huge popularity in the last few years. Their simplicity allows for very large scale training and relatively fast testing, as they can be implemented in terms of convolutions. They can also reach state-of-the-art performance provided one use discriminant enough features. Indeed, such systems have ranked atop of the popular Pascal VOC detection challenge from 2006 to 2011 (Everingham et al., 2006, 2007, 2008, 2009, 2010, 2011). Deformable Part Models (DPMs) (Felzenszwalb and Huttenlocher, 2005; Felzenszwalb et al., 2010b) (introduced formally in § 6.3) are the latest incarnations of such systems, and the winners of many past challenges.

The algorithm we propose leverages the classical use of the Fourier transform to accelerate the multiple evaluations of a linear predictor in a multi-scale sliding-window detection scheme. Despite relying on a classic result of signal processing, the practical implementation of this strategy is not straightforward and requires a careful organization of the computation. It can be summarized in three main ideas: (a) we exploit the linearity of the Fourier transform to avoid having one such transform per image feature (see § 4.3.2), (b) we control the memory usage required to store the transforms of the filters by building patchworks combining the multiple scales of an image (see § 4.4.1), and finally (c) we optimize the use of the processor cache by computing the Fourier domain point-wise multiplications in small fragments (see § 4.4.2).

4.2 Related works

Popular methods to search a large space of candidate object locations include cascades of simple classifiers (Viola and Jones, 2001), salient regions (Perko and Leonardis, 2007), Hough transform based detection methods (Maji and Malik, 2009), or branch-and-bound (Lampert et al., 2008). Regarding Deformable Part Models (DPMs) specifically, a selection of the most relevant works aiming at making them faster include (Felzenszwalb et al., 2010a), which sees the parts as classifiers in a cascade, and splits the detection process into two passes. The first pass evaluates the detector on a low-dimensional feature space (reduced from 32 HOG features to 5 using PCA), and the second pass with all the features. Expressing the part filters as a sparse linear combination of a dictionary, (Pirsiavash and Ramanan, 2012) can obtain large speedups when detecting multiple objects simultaneously, since in this case the dictionary typically can be made much smaller than the total number of filters while not sacrificing too much accuracy, as many objects share visually similar parts, *e.g.* wheels, limbs, corners, *etc.* Finally (Kokkinos, 2011) applies the dual-tree branch and bound algorithm (Lampert et al.,

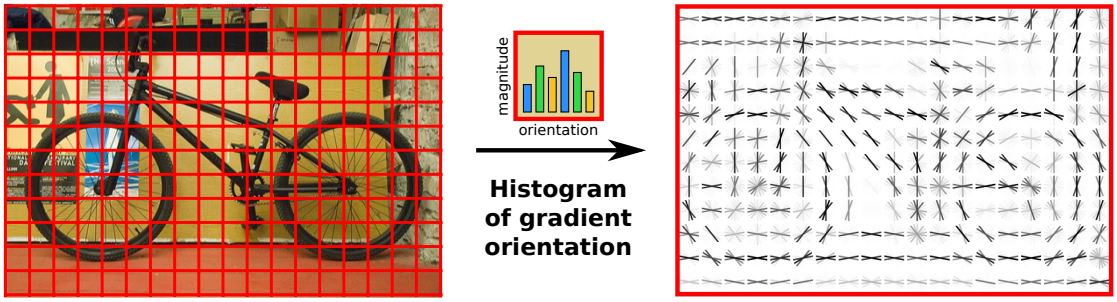


Figure 4.1 – Representation of the computation of the Histogram of Oriented Gradients (HOG) features (Dalal and Triggs, 2005). It is common to repeat the process with multiple resized version of the input image in order to detect object of different sizes with the same detector.

2009) to more efficiently optimize the objective function of (Felzenszwalb et al., 2010a), and rapidly approximates the inner products between filters and HOG features by quantizing the HOG cells onto a codebook and replacing their inner products with lookups of precomputed scores in (Kokkinos, 2012).

All these methods have in common that they are approximate, with no guaranteed speedup in the worst case. Cascades in particular are also notoriously hard to tune in order to obtain good performance without sacrificing too much accuracy, often requiring a dedicated validation set (Zhang and Viola, 2007; Viola and Jones, 2001). The approach we pursue here is akin to (Cecotti and Graeser, 2008), taking advantage of properties of the Fourier transform to speed up linear object detectors using multiple features while remaining exact.

Besides accelerating the evaluation of the detector at each possible location, other works have already dealt with the problem of the efficient computation of the feature pyramid and, in the case of DPMs, of the optimal assignment of the parts' locations. The fast construction of the complete image pyramid and associated features computation at each scale has been addressed by (Dollar et al., 2010). Their idea is to compute such features only once per octave and interpolate the scales in-between, making the whole process typically an order of magnitude faster with only a minor loss in detection accuracy. (Felzenszwalb and Huttenlocher, 2004) provides linear time algorithms for solving maximization problems involving an arbitrary sampled function and a spatial quadratic cost. By using deformation costs of this form, the optimal assignment of the parts' locations can be efficiently computed.

4.3 Linear object detectors and Fourier transform

Many linear object detectors – such as the HOG detector of Dalal and Triggs (Dalal and Triggs, 2005); the Deformable Part Model (DPM) of Felzenszwalb *et al.* (Felzenszwalb and Huttenlocher, 2005; Felzenszwalb et al., 2010b); or the Convolutional Neural Network (CNN) of LeCun *et al.* (LeCun et al., 1998a) – extract features densely from the image of interest (as well as the outputs of the previous layers in the case of CNNs) before evaluating the model.

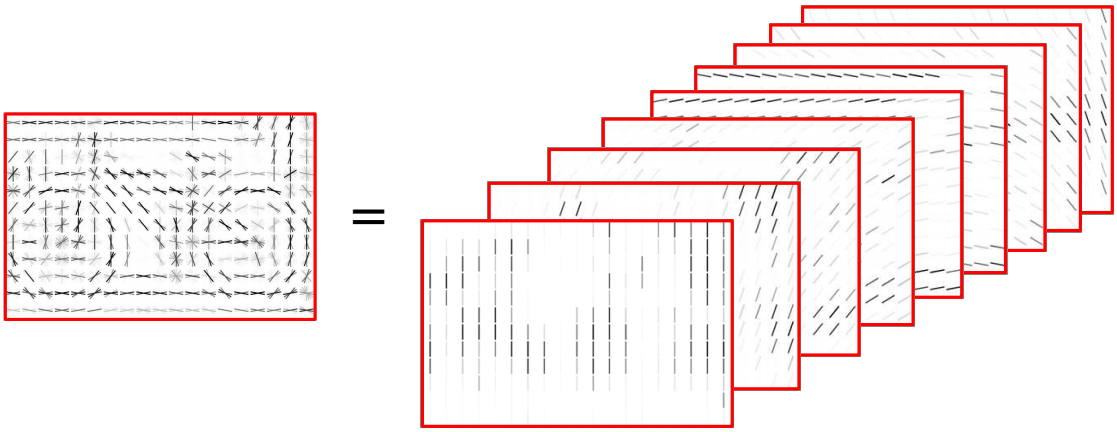


Figure 4.2 – Alternative view of the HOG features, organized in planes containing distinct features instead of a grid. The filters trained by the detector are similar in composition.

In order to make the detector scale invariant, these features are often extracted from every scale of a standard image pyramid, produced via repeated smoothing and subsampling of the input image. One can imagine these features as being arranged on a coarse grid with several features extracted from each grid cell. For example, the Histogram of Oriented Gradients (HOG) features (Dalal and Triggs, 2005) which we use in all our experiments correspond to the bins of an histogram of the gradient orientations of the pixels within the cell, as represented in figure 4.1. They use cells of size typically 8×8 pixels (Dalal and Triggs, 2005; Felzenszwalb and Huttenlocher, 2005), with a few dozen distinct features per cell (36 in (Dalal and Triggs, 2005); 32 in (Felzenszwalb et al., 2010b, 2011) which we use as a baseline). An alternative description of the arrangement of the features is to view them as organized in *planes* as depicted in figure 4.2. These planes are analogous to the *RGB* channels of standard color images, but instead of colors they each contain a distinct feature from each cell of the grid. The filters trained by the detector are similar in composition, containing the exact same number of feature planes.

4.3.1 Evaluation of a linear detector as a convolution

Let K stands for the number of features, $\mathbf{x}_k \in \mathbb{R}^{M \times N}$ for the k^{th} feature plane extracted from a particular image, and $\mathbf{w}_k \in \mathbb{R}^{P \times Q}$ for the k^{th} feature plane of a particular filter. The scores $\mathbf{z} \in \mathbb{R}^{(M-P+1)\times(N-Q+1)}$ of a filter evaluated on an image are given by the following formula:

$$z(i,\, j) = \sum_k \sum_p \sum_q x_k(i+p-1,\, j+q-1)\, w_k(p,\, q) \tag{4.1}$$

that is the sum across feature planes of the Frobenius inner products of the features extracted from the sub-window of size $P \times Q$ anchored at position (i, j) and the filter. Computing the scores of a filter at all possible locations can thus be done by summing across features the

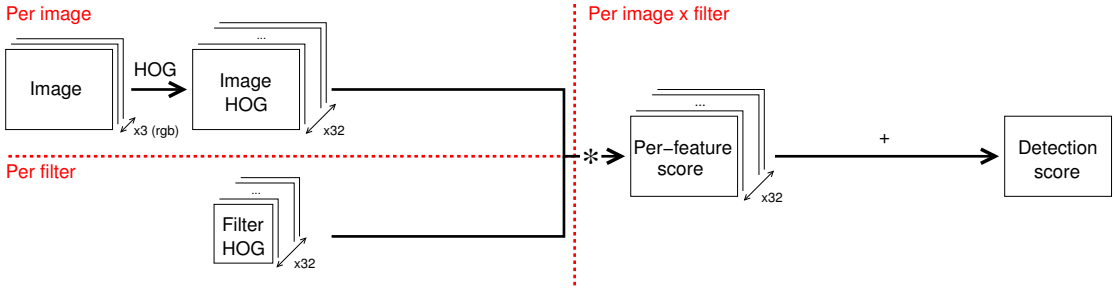


Figure 4.3 – Standard convolution process, convolving and summing all image and filter planes.

results of the convolutions of the image and the (reversed) filter, i.e.

$$\mathbf{z} = \sum_k \mathbf{x}_k * \bar{\mathbf{w}}_k \tag{4.2}$$

where $\bar{\mathbf{w}}$ is the reversed filter ($\bar{w}_k(i,j) = w_k(P-i+1, Q-j+1)$).

The computational cost of a standard convolution between an image of size $M \times N$ and a filter of size $P \times Q$ is in $\Theta(MNPQ)$. More precisely the number of floating point operations is

$$C_{\text{std}} = 2(M-P+1)(N-P+1)PQ \tag{4.3}$$

corresponding to one multiplication and one addition for each score and each filter coefficient. Ultimately one needs to convolve L filters and sum them across K feature planes (see figure 4.3), bringing the total number of operations per image to

$$C_{\text{std/image}} = KLC_{\text{std}}. \tag{4.4}$$

4.3.2 Leveraging the Fourier transform

It is well known that convolving in the original signal space is equivalent to point-wise multiplying in the Fourier domain. Convolutions done by first computing the Fourier transforms of the input signals, multiplying them in Fourier domain, before taking the inverse transform of the result can also be more efficient if the filter size is large enough. Indeed, the computational cost of a convolution done with the help of the Fourier Transform is $\Theta(MN \log MN)$.

If we define

$$C_{\text{FFT}} \approx 2.5MN \log_2 MN \tag{4.5}$$

$$C_{\text{mul}} = 4MN \tag{4.6}$$

the costs of one Fourier transform (the approximation of C_{FFT} comes from (Frigo and Johnson,

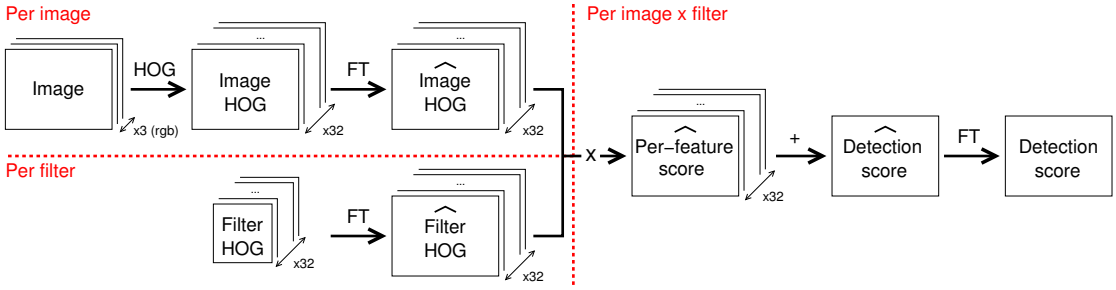


Figure 4.4 – Fast Fourier convolution process taking advantage of the fact that the inverse Fourier transform that produces the final detection score needs to be done only once per image / filter pair, and not once per feature, since the sum across planes can be done in the Fourier domain.

2005)) and of the point-wise multiplications respectively (one complex multiplication and one complex addition, halved because of the symmetry of the transform of a real signal), the total cost is

$$C_{\text{Fourier}} = 3\,C_{\text{FFT}} + C_{\text{mul}} \tag{4.7}$$

per product for the three (two forward and one inverse) transforms using a Fast Fourier Transform (FFT) algorithm (Frigo and Johnson, 2005). Note that the filters' forward Fourier transforms can be done off-line, and thus should not be counted in the overall detection time, and that an image's forward Fourier transform has to be done only once, independently of the number of filters. Moreover, in the case of learning methods based on bootstrapping examples, the images' forward Fourier transforms can also be done offline for training.

Taking all this into account, and using the linearity property of the Fourier transform, one can drastically reduce the cost per image from KLC_{Fourier}. Since the Fourier transform is linear, it does not matter if the sum across planes is done before or after the inverse transforms. If done before, only one inverse transform per filter will be needed even if there are multiple planes. Together with the fact that the forward transforms need to be done only once per filter or per image (see figure 4.4), the total cost per image is

$$C_{\text{Fourier/image}} = \underbrace{KC_{\text{FFT}}}_{\text{forward FFTs}} + \underbrace{KLC_{\text{mul}}}_{\text{multiplications}} + \underbrace{LC_{\text{FFT}}}_{\text{inverse FFTs}} \tag{4.8}$$

enabling large computational gains if $K + L \ll KL$ and/or $P, Q \gg 1$.

Plugging in typical numbers ($M, N = 64$, $P, Q = 6$, $K = 32$, $L = 54$ as in (Felzenszwalb et al., 2011)), doing the convolutions with Fourier results in a theoretical speedup factor of 11 compared to the standard convolution process. Taking into account the fact that the features are frequently padded with zeroes on all sides to allow the filters to partially go out of the image, and that the Fourier transform needs only half of the amount of padding since it computes circular convolutions, this speedup can even be greater (14 in the case where the padding is

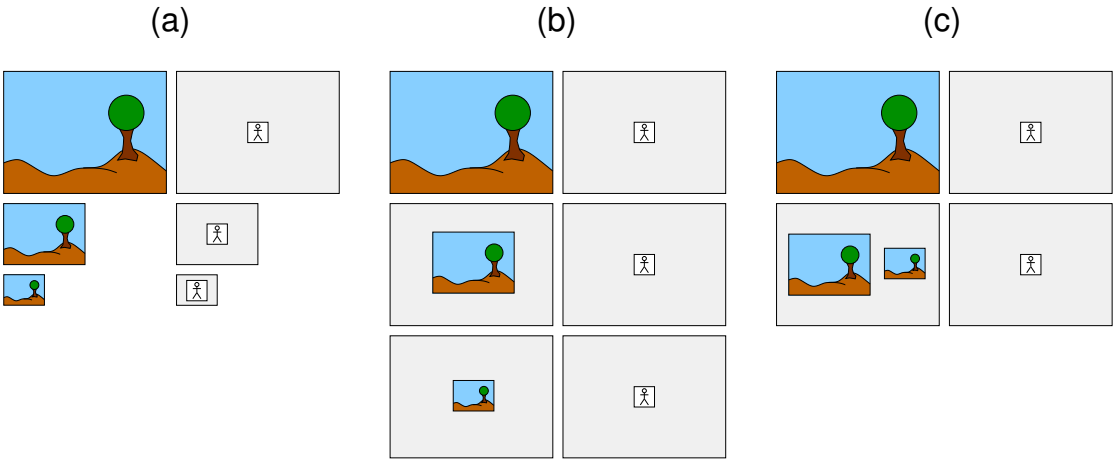


Figure 4.5 – The computation of the point-wise multiplications between the Fourier transform of an image and that of a filter requires to pad them to the same size before computing the transforms. Given that images have to be parsed at multiple scales, the naive strategy is either to store for each filter the transforms corresponding to a variety of sizes (a), or to store only one version of the filter's transform and to pad the multiple scales of each image (b). Both of these strategies are unsatisfactory either in term of memory consumption (a) or computational cost (b). We propose instead a patchwork approach (c), which consists of creating patchwork images composed of the multiple scales of the image, and has the advantages of both alternatives.

equal to the filter size). This cost is independent of the filter size $P \times Q$, resulting in even larger gains for larger filters. The FFT also has excellent numerical properties (Schatzman, 1996), as demonstrated by our experiments. There is no precision loss for small filter sizes, and an increase in precision for larger ones. Finally, one can also reduce by half the cost of computing the Fourier transforms of the filters if they are symmetric (Felzenszwalb et al., 2010b), or come by symmetric pairs (Felzenszwalb et al., 2011).

4.4 Implementation strategies

Implementing the convolutions with the help of the Fourier transform is straightforward, but involves two difficulties: memory over-consumption and lack of memory bandwidth. These two problems can be remedied using methods presented in the following subsections.

4.4.1 Patchworks of pyramid scales

The computational cost analysis of § 4.3.2 was done under the assumption that the Fourier transforms of the filters were already precomputed. But the computation of the point-wise multiplications between the Fourier transforms of an image and that of a filter requires to first pad them to the same size. Images can be of various sizes and aspect-ratio, especially since they are parsed at multiple scales, and precomputing filters at all possible sizes as in

Table 4.1 – Asymptotic memory footprint and computational cost for the three approaches described in § 4.4.1, to process one image of size $M \times N$ with L filters, at scales 1, ρ, ρ^2, … The factor $\frac{1}{1-\rho^2} = \sum_{k=0}^{+\infty} \rho^{2k}$ accounts for the multiple scales of the image pyramid, while $\frac{\log MN}{1-\rho^2} \approx -\frac{\log MN}{\log \rho^2}$ for $\rho \approx 1$ is the number of scales to visit. Taking the same typical values as in § 4.3.2 for $M, N = 64$, $L = 54$, and $\rho = 0.9$ gives $\frac{1}{1-\rho^2} \approx 5.3$ and $\frac{\log MN}{1-\rho^2} \approx 44$. Our patchwork method (c) combines the advantages of both methods (a) and (b).

Approach	Memory (image + filters)					Computational cost				
(a)	$\frac{1}{1-\rho^2} MN$	+	$\frac{1}{1-\rho^2} LMN$	$\approx$	$1.2 \cdot 10^6$	$\frac{1}{1-\rho^2} LMN$	$\approx$	$1.2 \cdot 10^6$
(b)	$-\frac{\log MN}{1-\rho^2} MN$	+	LMN	$\approx$	$3.8 \cdot 10^5$	$-\frac{\log MN}{1-\rho^2} LMN$	$\approx$	$8.7 \cdot 10^6$
(c)	$\frac{1}{1-\rho^2} MN$	+	LMN	$\approx$	$2.4 \cdot 10^5$	$\frac{1}{1-\rho^2} LMN$	$\approx$	$1.2 \cdot 10^6$

figure 4.5(a) might be unrealistic in term of memory consumption. Another approach could be to precompute the transforms of the images and the filters padded only to the largest image size, as shown in figure 4.5(b). This would require as little memory as possible for the filters, but would result in an additional computational burden to compute the transforms of the images, and more importantly to perform the point-wise multiplications.

However, a simpler and more efficient approach exists, combining the advantages of both alternatives. By grouping images together in patchworks of the size of the largest image, one needs to compute the transforms of the filters only at that size, while the amount of padding needed is much less than required by the second approach. We observed it experimentally to be less than 20%, vs. 87% for the second approach. The performance thus stays competitive with the first approach while retaining the memory footprint of the second (see table 4.1 for an asymptotical analysis). The grouping of the images does not need to be optimal, and fast heuristics exist, such as the bottom-left bin-packing heuristic (Chazelle, 1983).

4.4.2 Taking advantage of the cache

A naive implementation of the main computation, that is the point-wise multiplications between the patchworks' Fourier transforms and the filters' Fourier transforms would simply loop over all patchworks and all filters. This would require to reload both from memory for each pairwise product as they are likely too large to all fit in the CPU cache, a small but very fast memory integrated to the CPU to reduce the access time and increase the bandwidth of frequently used data from the main memory.

We observed in practice that such an implementation is indeed limited by the speed and bandwidth of the main memory. However, reorganizing the computation allows to remove this bottleneck. Let R be the total number of patchworks to process, L the number of filters, K the number of features, $M \times N$ the size of the patchworks' F transforms, $u(F)$ the time it takes to point-wise multiply together two planes of F coefficients, and $v(F)$ the time it takes to read

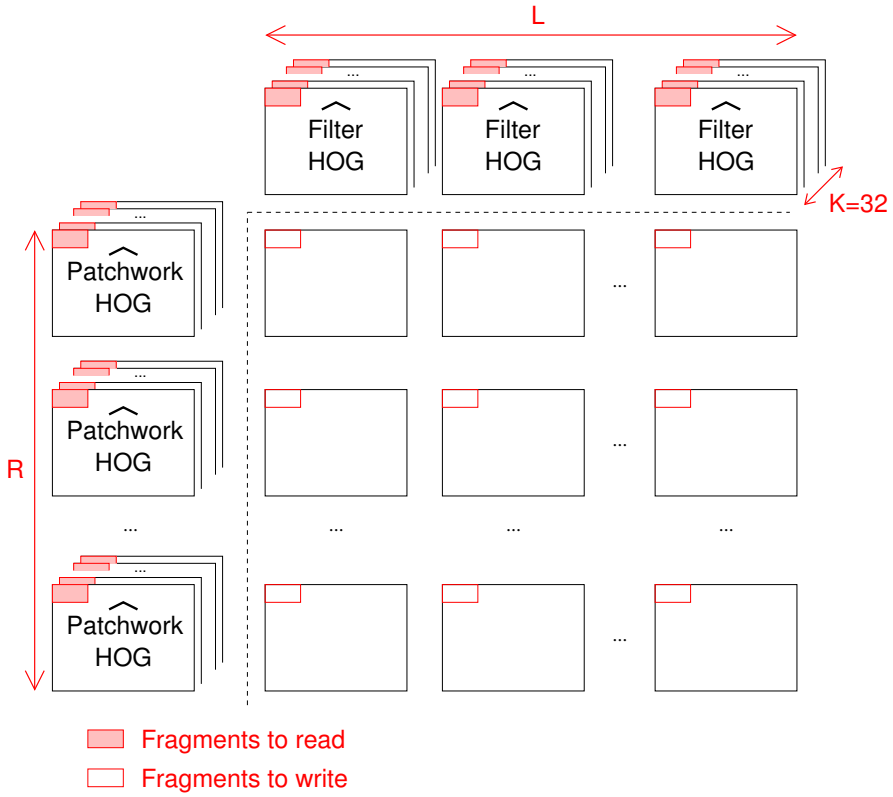


Figure 4.6 – To compute the point-wise products between each of the Fourier transforms of the R patchworks, and each of the transforms of the L filters, the naive procedure loops through every pair. This strategy unfortunately requires multiple CPU cache violations, since the transforms are likely to be too large to all fit in cache, resulting in a slow computation of each one of the LR products. We propose instead to decompose the transforms into fragments (here shown as red rectangles), and to have an outer loop through them. With such a strategy, by loading a total of $L+R$ fragments in the CPU cache, we end up computing LR point-wise products between fragments.

(resp. write) F coefficients from (resp. into) the memory to (resp. from) the CPU cache.

A naive strategy going through every patchwork / filter pair results in a total processing time of

$$T_{\text{naive}} = \underbrace{KLR2v(MN)}_{\text{reading}} + \underbrace{KLRu(MN)}_{\text{multiplications}} + \underbrace{LRv(MN)}_{\text{writing}}. \tag{4.9}$$

This is mainly due to the bad use of the cache, which is constantly reloaded with new data from the main memory.

We can improve this strategy by decomposing transforms into *fragments* of size F, and by adding an outer loop through these $\frac{MN}{F}$ fragments (see figure 4.6 and algorithm 4.1). The

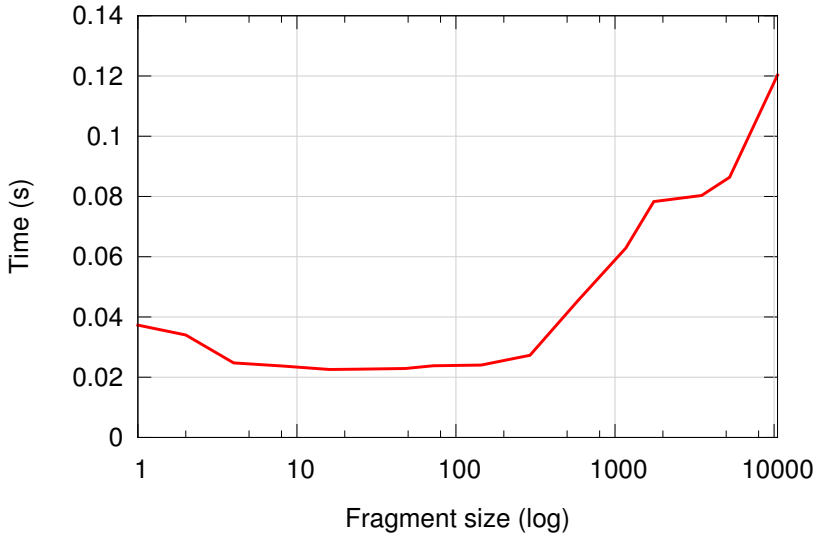


Figure 4.7 – Average time taken by the point-wise multiplications (in seconds) for different fragment sizes (number of coefficients) for one image of the Pascal VOC 2007 challenge.

cache usage will be $K(R+1)F$, and the time to process all patchwork / filter pairs will become

$$T_{\text{fast}} = \underbrace{\frac{MN}{F}}_{\text{number of fragments}} \left(\underbrace{K(L+R)\,v(F)}_{\text{reading}} + \underbrace{KLR\,u(F)}_{\text{multiplications}} + \underbrace{LR\,v(F)}_{\text{writing}} \right) \tag{4.10}$$

$$= \underbrace{K(L+R)\,v(MN)}_{\text{reading}} + \underbrace{KLR\,u(MN)}_{\text{multiplications}} + \underbrace{LR\,v(MN)}_{\text{writing}}. \tag{4.11}$$

By making F small, we could reduce the cache usage arbitrarily. However, CPUs are able to load from the main memory in bursts, which makes values smaller than that burst size sub-optimal (see figure 4.7). The speed ratio between the naive and the fast methods is

$$\frac{T_{\text{naive}}}{T_{\text{fast}}} = \frac{(2+\frac{1}{K}) + \frac{u(MN)}{v(MN)}}{(\frac{L+R}{LR} + \frac{1}{K}) + \frac{u(MN)}{v(MN)}} \approx 2\frac{v(MN)}{u(MN)} + 1. \tag{4.12}$$

In practice, the cache can hold at least one patchwork of size MN and the actual speedup we observe is around 5.7. Decomposing the transforms into fragments also scales better across multiple CPU cores, as they can focus on distinct parts of the transforms, instead of all loading the same patchwork or filter.

Algorithm 4.1 Fast Fourier convolution process taking as input the pyramid levels already packed into R patchworks and the precomputed L filters already reversed, padded to the patchwork size, and Fourier transformed.

Input: patchworks $\mathbf{x}_k^r$, transformed filters $\hat{\mathbf{w}}_k^l$
  for $r \leftarrow 1, \ldots, R$ **do** `# Iterate over the patchworks`
    for $k \leftarrow 1, \ldots, K$ **do** `# Iterate over the features`
      $\hat{\mathbf{x}}_k^r \leftarrow \mathrm{FFT}\left(\mathbf{x}_k^r\right)$
    end for
  end for
  for $i \leftarrow 1, \ldots, MN$ **do** `# Iterate over the fragments`
    for $r \leftarrow 1, \ldots, R$ **do** `# Iterate over the patchworks`
      for $l \leftarrow 1, \ldots, L$ **do** `# Iterate over the filters`
        $\hat{\mathbf{z}}_r^l(i) \leftarrow \sum_{k=1}^{K} \hat{\mathbf{x}}_k^r(i) \cdot \hat{\mathbf{w}}_k^l(i)$ `# Sum across features`
      end for
    end for
  end for
Output: $\mathbf{z}_r^l \leftarrow \mathrm{FFT}^{-1}\left(\hat{\mathbf{z}}_r^l\right)$ `# Return the scores`

4.5 Experiments

To evaluate our approach for linear object detector acceleration we compared it to the publicly available system from (Felzenszwalb et al., 2011). We used the models already present in the system, trained on the Pascal VOC 2007 challenge (Everingham et al., 2007) dataset, which achieve close to state-of-the-art detection results. Note that (Felzenszwalb et al., 2011) provides several implementations of the convolutions, ranging from the most basic to the most heavily optimized.

The evaluation was done over all 20 classes of the challenge by looking at the detection time speedup with respect to the fastest baseline convolution implementation on the same machine. The baseline is written in assembly and makes use of both CPU SIMD instructions and multi-threading. As our method is exact, the average precision should stay the same up to numerical precision issues. The results are given in table 4.2 for verification purposes. The small discrepancy compared to the results of (Felzenszwalb et al., 2011) might be explained by the fact that we did not use the provided code to resize the images when building the feature pyramids.

We used the *FFTW* (version 3.3) library (Frigo and Johnson, 2005) to compute the FFTs, and the *Eigen* (version 3.0) library (Guennebaud et al., 2010) for the remaining linear algebra. Both libraries are very fast as they make use of the CPU SIMD instruction sets. Our experiments timed in the same conditions on the same 2.2 GHz Intel Core i7 Quad machine show that our approach achieves a significant speedup, being more than seven times faster (see table 4.3). We compare only the time taken by the convolutions in order to be fair to the baseline, some

Table 4.2 – Pascal VOC 2007 challenge results.

	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table
V4 (%)	28.9	59.5	10.0	15.2	25.5	49.6	57.9	19.3	22.4	25.2	23.3
Ours (%)	29.4	58.9	10.0	13.4	25.3	50.6	57.6	18.9	22.6	24.9	24.4

	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mean
V4 (%)	11.1	56.8	48.7	41.9	12.2	17.8	33.6	45.1	41.6	32.3
Ours (%)	11.5	56.7	47.3	42.4	13.0	19.2	34.8	46.3	40.4	32.4

of its other components being written in Matlab, while our implementation is written fully in C++. The average time taken by the baseline implementation to convolve a feature pyramid (10 scales per octave) with all the filters of a particular class (54 filters, most of them of size 6×6) was 413 ms. The average time taken by our implementation was 56 ms, including the forward FFTs of the images. For comparison, the time taken in our implementation to compute the HOG features (including loading and resizing the image) was on average 64 ms, while the time taken by the distance transforms was 42 ms, the time taken by the remaining components of the system being negligible.

We also tested the numerical precision of both approaches. The maximum absolute difference that we observed between the baseline and a more precise implementation (using double precision) was 9.5×10^{-7}, while for our approach it was 4.8×10^{-7}. The mean absolute difference were respectively 2.4×10^{-8} and 1.8×10^{-8}.

While the speed and numerical precision of the baseline degrade proportionally with the filter size, they remain constant with our approach, enabling the use of larger filters for free. For example if one were to use filters of size 8×8 instead of 6×6 as in many of the current models, the advantage of our method over the baseline would increase by a factor $\frac{8 \times 8}{6 \times 6} \approx 1.8$.

4.6 Conclusion

The idea motivating our work is that the Fourier transform is linear, enabling one to do the addition of the convolutions across feature planes in Fourier space, and be left in the end with only one inverse Fourier transform to do. To take advantage of this, we proposed two additional implementation strategies, ensuring maximum efficiency without requiring huge memory space and/or bandwidth, and thus making the whole approach practical.

The method increases the speed of many state-of-the-art object detectors severalfold with no loss in accuracy when using small filters, and becomes even faster and more accurate with larger ones. That such an approach is possible is not entirely trivial (the reference implementation of (Felzenszwalb et al., 2011) contains five different ways to do the convolutions,

Table 4.3 – Pascal VOC 2007 challenge convolution time and speedup.

	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table
V4 (ms)	409	437	403	414	366	439	352	432	417	429	450
Ours (ms)	55	56	53	56	57	56	54	56	56	57	57
Speedup (x)	7.4	7.8	7.6	7.4	6.4	7.9	6.5	7.7	7.5	7.5	8.0

	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mean
V4 (ms)	445	439	429	379	358	351	425	458	433	413
Ours (ms)	57	59	57	54	54	55	57	58	55	56
Speedup (x)	7.8	7.5	7.6	7.0	6.6	6.4	7.4	7.9	7.9	7.4

all at least an order of magnitude slower); nevertheless, the analysis we developed is readily applicable to many other systems, such as Convolutional Neural Networks.

5 Accelerated Training of Linear Object Detectors

In this chapter we present our contributions to speed up the training of a broad range of linear object detectors. Our approach consists of reformulating the computation of the gradients as a convolution, and to use the same strategies as in the previous chapter to accelerate it. We obtain a speedup factor proportional to the filter size without relying on the sparsity induced by a specific loss, nor on a stochastic sub-sampling of the training examples. Content presented in this chapter is based on the following publication (Dubout and Fleuret, 2013a):

> C. Dubout and F. Fleuret. Accelerated Training of Linear Object Detectors. In *CVPR 2013 Workshop on Structured Prediction*, 2013.

5.1 Introduction and related Works

Linear object detectors are typically used in a sliding-window fashion, predicting a score related to the presence or absence of an object for each possible position and scale in the scene to process. These scores are computed by taking the inner product between the corresponding image sub-windows and the classifier weights. It is straightforward to see that the entire score matrix can be computed by taking the convolution of the image with the (reversed) linear filter corresponding to the learned classifier weights. Sophisticated methods such as Deformable Part Models (DPMs) (Felzenszwalb and Huttenlocher, 2005; Felzenszwalb et al., 2010b) combine multiple such detectors, either in mixtures, and/or in multi-part models.

Given a loss which has the form of a sum over all locations and scales of a per-sample loss, we can similarly reformulate the value of its gradient as a convolution. As we show in § 5.2, it is the convolution of the map of point-wise derivatives of the loss – that is, at each point, how the loss changes when the response of the predictor changes there – with the map of feature responses.

By leveraging this form, the computation of the gradient can be sped up by using Fourier transforms, exploiting the redundancy between overlapping samples, as revealed by the analysis of § 5.3. In practice, as demonstrated in § 5.4, such organization of the computation removes the increase of the cost with the size of the filters, which are always smaller than the scene to process.

A large amount of literature deals with the problem of efficiently training linear classifiers, such as linear SVMs, by exploiting the particular nature of the associated loss function, novel convex optimization algorithms, or clever implementation strategies (Platt, 1999; Joachims, 2006; Shalev-Shwartz et al., 2007; Fan et al., 2008; Hsieh et al., 2008). We follow an orthogonal approach extending the one of the previous chapter and look for algorithmic gains in the specific case where the training examples are overlapping sub-windows extracted from training images. As most computer vision learning problems involve very large training sets, a consensus in the vision community is to use online or stochastic gradient descent algorithms (Bottou and LeCun, 2003; Felzenszwalb et al., 2010b; Wijnhoven and de With, 2010).

5.2 Evaluation of the gradient of a linear detector as a convolution

As in the previous chapter, we handle the parsing at multiple scales by considering that we process *patchworks*, each composed of multiple scales of one of the original images of the training set. In the rest of the article, an *image* can refer to either one of the original images of the training set, or one of these constructed patchworks.

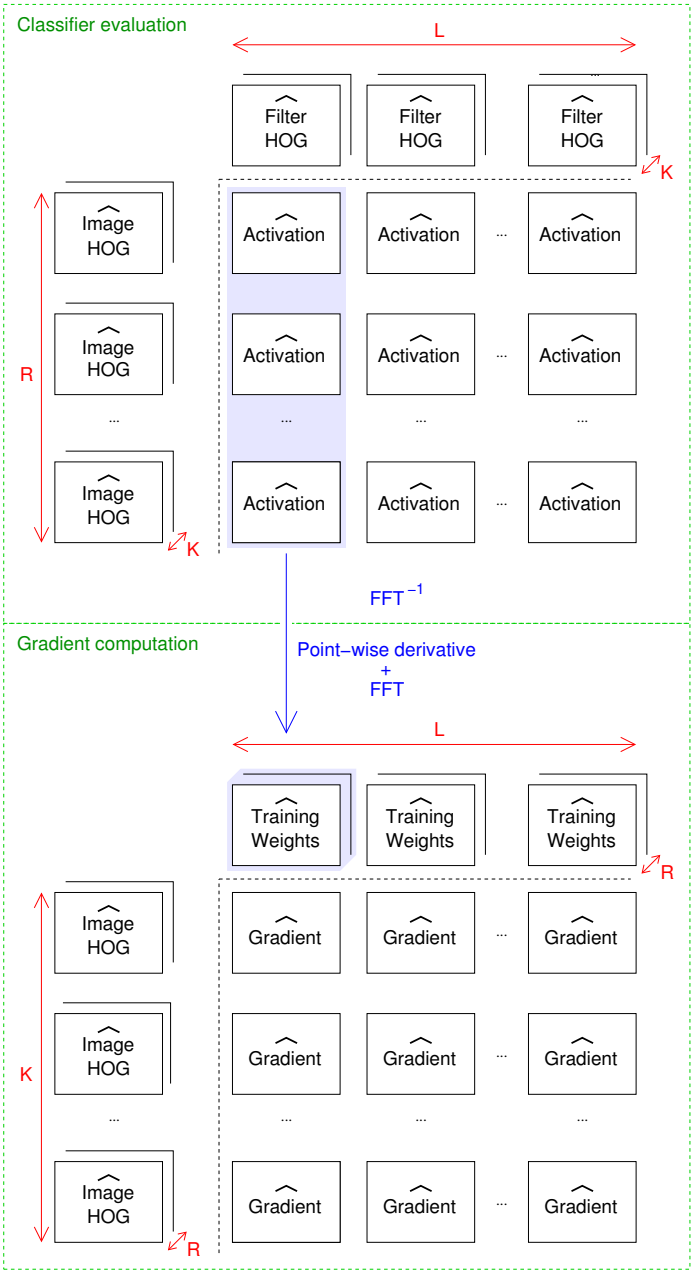


Figure 5.1 – Computation of the gradient of the loss with respect to the model weights, in the case of R images, K features, and L linear filters. The top figure depicts the computation of the previous chapter, and consists of the series of point-wise products between the Fourier transforms of the filters, and the Fourier transforms of the images, followed by the inverse Fourier transforms. This produces the maps of point-wise evaluations of the detector in each image. The bottom figure depicts the computation of the gradient. It first computes the point-wise derivatives of the loss to obtain the point-wise training weights, and then the Fourier transforms of the obtained maps. Then, for each feature and each filter, the Fourier transform of the gradient of the loss is obtained by summing the point-wise products of the R training weight maps with the R image maps for that feature.

In the case of a single linear predictor for object detection and a single image, the loss function minimized during the training of the detector typically has a data-driven term of the form

$$L(\mathbf{w}) = \sum_i \sum_j l\left(y(i,\,j)z(i,\,j)\right) \tag{5.1}$$

where $\mathbf{y} \in \{-1,\,1\}^{(M-P+1)\times(N-Q+1)}$ are the labels of the sub-images corresponding one-to-one to the sub-images of the scores $\mathbf{z}$ and l is the loss per sample. This expression extends naturally to multiple filters and multiple images by adding sums over them.

From this, we derive for each feature plane k of the filter $\mathbf{w}$

$$\begin{aligned}
\nabla L_k(p,\,q) &= \frac{\partial L(\mathbf{w}_k)}{\partial w_k(p,\,q)} && (5.2)\\
&= \sum_i \sum_j \frac{\partial l\left(y(i,\,j)z(i,\,j)\right)}{\partial w_k(p,\,q)} && (5.3)\\
&= \sum_i \sum_j y(i,\,j)\, l'\left(y(i,\,j)z(i,\,j)\right) \frac{\partial z(i,\,j)}{\partial w_k(p,\,q)} && (5.4)\\
&= \sum_i \sum_j \left(\mathbf{y}\cdot l'\left(\mathbf{y}\cdot\mathbf{z}\right)\right)(i,\,j)\, x_k(i+p-1,\,j+q-1) && (5.5)\\
&= \left(\overline{\mathbf{y}\cdot l'\left(\mathbf{y}\cdot\mathbf{z}\right)} * \mathbf{x}_k\right)(p,\,q) && (5.6)
\end{aligned}$$

hence our main result

$$\nabla L_k = \overline{\mathbf{y}\cdot l'\left(\mathbf{y}\cdot\mathbf{z}\right)} * \mathbf{x}_k \tag{5.7}$$

where the operator $\cdot$ stands for the point-wise multiplication of two matrices and l' for both the loss derivative and its point-wise evaluation. This point-wise derivative can be interpreted as the signed sample weights, since it quantifies the importance of the sample at that location of the image in the change of the filter weights.

5.3 Computational cost of the gradient computation

We analyze the asymptotic costs of the gradient of the loss, which together with the computation of the scores of § 4.3.2 usually constitute most of the computational effort. These two computational steps are depicted on figure 5.1.

The cost of computing the gradient of the loss over one image for the standard method using (5.4) is in $\Theta(KLMNPQ)$, same as the cost required to compute the scores in § 4.3.1. Leveraging the Fourier transform to compute it as a convolution, as highlighted in (5.7), it can be reduced by realizing that the transform of the point-wise derivatives of the left-hand side of the convolution operator, i.e. $\overline{\mathbf{y}\cdot l'\left(\mathbf{y}\cdot\mathbf{z}\right)}$, does not depend on k and therefore can be shared across features. The cost to compute the point-wise derivatives themselves is negligible, as it

Algorithm 5.1 Our Fourier-based stochastic gradient descent algorithm, inspired from the *Pegasos* algorithm (Shalev-Shwartz et al., 2007), taking a whole scene as mini-batch. It minimizes a loss of the form $L(\mathbf{w}) = \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{RMN}\sum_r \sum_{i,j} l(y_r(i,j) f_r(i,j))$.

Input: λ, T
  $\hat{\mathbf{w}}_1 \leftarrow \mathbf{0}$ `# Initialize the filter to zero`
  for $t \leftarrow 1, \ldots, T$ **do**
    $r \leftarrow rand(1, \ldots, R)$ `# Pick a scene at random`
    $\hat{\mathbf{x}}_r \leftarrow \mathrm{FFT}(\mathbf{x}_r)$ `# Transform the scene`
    $\hat{\mathbf{f}} \leftarrow \hat{\mathbf{w}}_t \cdot \hat{\mathbf{x}}_r$ `# Convolve it with the current filter`
    $\mathbf{f} \leftarrow \mathrm{FFT}^{-1}(\hat{\mathbf{f}})$ `# Get back the scores`
    $\hat{\mathbf{y}} \leftarrow \mathrm{FFT}(\mathbf{y}_r \, l'(\mathbf{y}_r \cdot \mathbf{f}))$ `# Transform the derivatives`
    $\eta_t \leftarrow \frac{1}{\lambda t}$ `# Current learning rate`
    $\hat{\mathbf{w}}_{t+1} \leftarrow (1 - \eta_t \lambda)\hat{\mathbf{w}}_t + \frac{\eta_t}{MN}\left(\hat{\mathbf{x}}_r \cdot \hat{\mathbf{y}}\right)$
    `# Update the filter with the gradient`
  end for
Output: $\mathbf{w} \leftarrow \mathrm{FFT}^{-1}(\hat{\mathbf{w}}_{T+1})$ `# Return the filter`

is in $\Theta(LMN)$. Assuming that the images were already transformed (already required in order to compute the scores), the cost to compute the gradient leveraging the FFT is

$$\underbrace{\Theta(LMN\log(MN))}_{\text{Forward FFTs of the derivatives}} + \underbrace{\Theta(KLMN)}_{\text{Multiplications}} \tag{5.8}$$

where the left term is the cost to transform the point-wise derivatives of each filter, and the right term is the cost of the convolutions. Since during training the filters are usually updated by adding them together with the gradients (scaled), one can typically keep them exclusively in Fourier space, removing the cost of transforming the filters back and forth. If it proves impossible, an additional $\Theta(KLMN\log(MN))$ term is required, reducing the gain compared to the standard process from $\Theta(PQ)$ to $\Theta\left(\frac{PQ}{\log(MN)}\right)$, but still keeping the total cost independent of the filter size.

As in the previous chapter, one can use the linearity of the Fourier transform to reduce the number of inverse transforms by summing this time across images in the frequency space. In that case, even if one has to transform back and forth the filters, the cost to compute the gradient over R images is R times that of (5.8) plus the optional transforms of the filters, in $\Theta(KLMN\log(MN))$, which for both K and R large enough (of order $\log(MN)$ or more), is $\Theta(RKLMN)$, again a gain by a factor $\Theta(PQ)$ compared to the standard process.

5.4 Experiments

To evaluate our approach to speed up the training of linear object detection systems, we trained a mixture of 6 filters, similar to the roots of the DPM of (Felzenszwalb et al., 2010b). Even though we only trained root filters, nothing prevents us from training full-fledged DPMs,

their loss consisting of a sum over part filters, which can be computed using our method, and a deformation penalty term, which can be handled separately at negligible cost.

We used the same modified Histogram of Oriented Gradients (HOG) features (Dalal and Triggs, 2005; Felzenszwalb et al., 2010b), the same initialization of the filters' positions, sizes, and left/right pose assignments as in (Felzenszwalb et al., 2011), and trained them on the PASCAL VOC 2007 challenge dataset (Everingham et al., 2007).

5.4.1 Implementation details

Typical computer vision datasets contain thousands of images, and thus potentially millions of (mostly negative) training examples, i.e. one per image sub-window at multiple scales. As recommended in (Felzenszwalb et al., 2010b; Bottou and LeCun, 2003) in such situations, we chose to train our classifier using a variant of the stochastic gradient descent algorithm. It is derived from the *Pegasos* algorithm (Shalev-Shwartz et al., 2007), using the Fourier transform to compute the convolutions, and without the projection step as it made no difference in our experiments. Since our method is efficient only at processing entire scenes, we took all the examples of a scene as a mini-batch at each stochastic gradient descent iteration. Algorithm 5.1 details the sketch of the algorithm.

We made some modifications to this algorithm in our experiments to adapt it to train a mixture model, and to improve its convergence speed as well as the quality of its final solution.

First as in (Felzenszwalb et al., 2011) we modified it to minimize the loss

$$L(\mathbf{w}) = \frac{\lambda}{2} \max_c ||\mathbf{w}^c||^2 + \frac{J}{N} \sum_{r,\, y^r(i,j)>0} \left| 1 - z^r_{y^r(i,j),\beta}(i,\, j) \right|^+ + \frac{1}{N} \sum_{r,\, y^r(i,j)=-1} \left| 1 + \max_c z^r_{c,\beta}(i,j) \right|^+ \quad (5.9)$$

where $\mathbf{w}^c$ is the filter of mixture component c, $y^r(i,\, j)$ is the index of the mixture associated to the sub-window anchored at $i,\, j$ in image r or -1 if it corresponds to a negative examples, J is a scale factor re-weighting the importance of the positive examples, N is the total number of examples taking J into account, $\lambda = \frac{1}{CN}$ is the regularization constant, and $|\cdot|^+ = \max(0, \cdot)$. $z^r_{c,\beta}$ is similar to the z of (4.1), except that it is computed for image r, mixture component filter $\mathbf{w}_c$, and a bias term β, i.e.

$$z^r_{c,\beta}(i,\, j) = \sum_k \sum_p \sum_q x^r_k(i+p-1,\, j+q-1)\, w^c_k(p,\, q) + \beta. \quad (5.10)$$

We pick the optimal bias β at the beginning of each stochastic gradient descent iteration in order to minimize the loss, i.e. $\beta_t = \operatorname{argmin}_\beta L(\mathbf{w}_t)$, as recommended in (Shalev-Shwartz et al., 2007) when dealing with large mini-batches. Since the bias β is identical among all mixture components, it does not influence their relative scores at test time, but we observed that it is of tremendous importance as removing it significantly reduces both the speed of the convergence of the algorithm and the quality of the final solution. At every iteration we

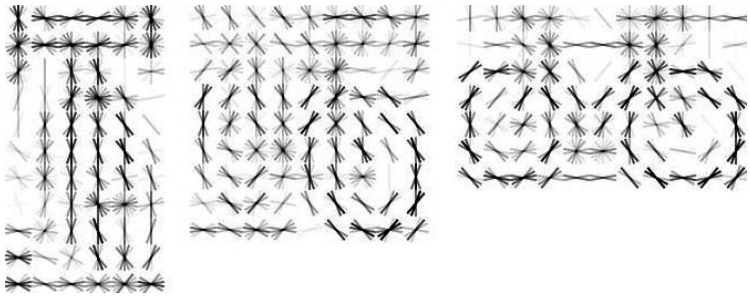


Figure 5.2 – Root filters for a bicycle model of normal size ((Felzenszwalb et al., 2010b)) learned on the PASCAL VOC 2007 dataset.

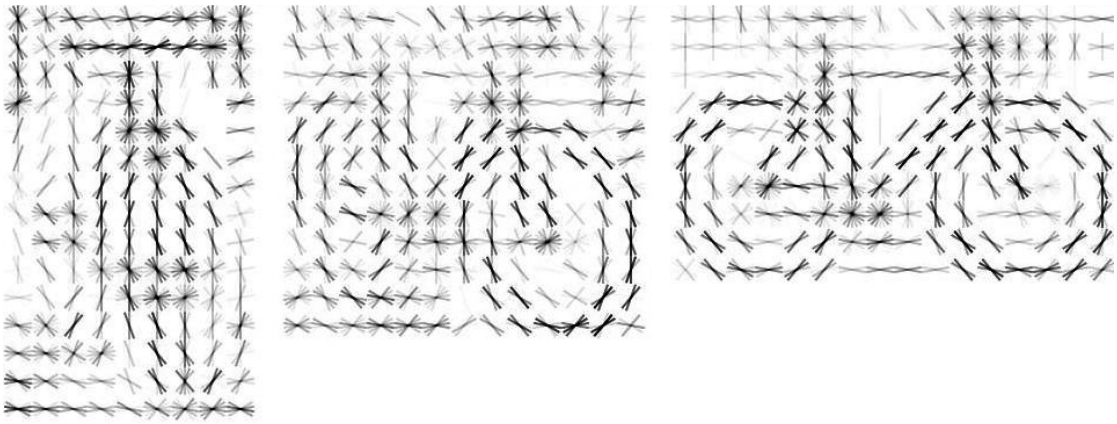


Figure 5.3 – Root filters for a bicycle model of double the normal size ((Felzenszwalb et al., 2010b)) learned on the PASCAL VOC 2007 dataset.

take all the negatives of a whole scene as a mini-batch, and add all the positive examples of the dataset to it. Considering all the positives at every iteration has a very small impact on the training time, the number of positives being usually very small even compared to the number of negatives of an unique scene, but improves drastically the convergence speed of the algorithm. The parameters we used were tuned on the provided validation set and were kept fixed in all experiments. They are $\lambda = 0.01$, $J = 5000$, and $T = 5000$.

5.4.2 Results

An example of a trained model is represented in figure 5.2. We also trained models twice as large as the size recommended in (Felzenszwalb et al., 2010b) for the root filters, and we show the same model, this time of twice the size in figure 5.3. The performances of those mixture models on all 20 classes of the PASCAL VOC challenge are displayed in table 5.1. We do not hope to compete with (Felzenszwalb et al., 2011), which trains more complex models including deformable parts, but only want to prove that our results are relevant with respect to the current state-of-the-art.

Table 5.1 – Average precision scores of the base system of VOC release 4 (Felzenszwalb et al., 2011), as well as our trained mixture on the PASCAL VOC 2007 challenge. As we trained only root filters, and not full part-based deformable models, we do not hope to compete with the V4 baseline. These results are provided only to demonstrate the relevance of our approach with respect to the state-of-the-art.

	aero	**bike**	**bird**	**boat**	**bottle**	**bus**	**car**	**cat**	**chair**	**cow**	**table**
V4 normal filters (%)	28.9	59.5	10.0	15.2	25.5	49.6	57.9	19.3	22.4	25.2	23.3
Ours normal filters (%)	18.2	40.8	4.2	11.1	15.0	24.7	34.0	4.7	11.5	27.9	10.7
Ours large filters (%)	18.4	47.3	2.5	13.1	16.9	29.1	41.2	10.3	12.5	26.7	11.2

	dog	**horse**	**mbike**	**person**	**plant**	**sheep**	**sofa**	**train**	**tv**	**mean**
V4 normal filters (%)	11.1	56.8	48.7	41.9	12.2	17.8	33.6	45.1	41.6	32.3
Ours normal filters (%)	5.2	27.7	34.2	18.5	10.8	18.5	12.9	27.1	20.6	18.9
Ours large filters (%)	5.7	37.4	32.6	22.5	11.4	19.3	18.4	24.8	22.9	21.2

Table 5.2 – Average time to compute the gradient of the loss for one stochastic gradient descent iteration. The standard sparse method relies on the sparsity of the samples weights induced by the hinge loss, and computes the gradient by visiting only the samples with non-zero weight. The acceleration it provides is strongly data-dependent.

	1 scene per batch		**10 scenes per batch**	
	Normal filters	**Large filters**	**Normal filters**	**Large filters**
Standard (ms)	41.3	70.9	390	699
Ours (ms)	7.2	7.4	33.1	33.1
Standard sparse (ms)	1.1	1.3	6.6	8.1

We implemented two versions of the gradient computation procedure, one using the standard method and one using the FFT, as detailed in § 5.2. Both versions make use of the CPU SIMD instruction sets as well as multi-threading. We timed their executions in the same conditions on the same 2.2 GHz Intel Core i7 Quad machine, and provide the results in table 5.2. We also tried to use larger mini-batches, processing 10 scenes together, which improves the advantage of our method over the generic one even more. Even though exploiting the sparsity of the loss was by far the fastest method in our experiment, this is due to the use of the hinge loss, and is strongly parameter (λ) and data dependent. The advantage of our method, as concluded from our analysis in § 5.3, is that it is faster without leveraging sparsity, and always take the same time, independently of the data, the loss, or the filter size. The time taken by the rest of the algorithm, mostly spent convolving the current scene with the filters is also independent of the size of the filters, and below 20ms per iteration. The algorithm typically converges to an acceptable solution in less than one epoch, which corresponds to 2 to 3 minutes.

5.5 Conclusion

We have presented a novel method to speed up the training of object detectors based on a linear classifier. Existing implementations of such methods relies on sparsity and sub-sampling of the training examples. Our approach by contrast, is based on a formulation of the gradient computation as a convolution, which allows to leverage the Fourier transform, and make the overall computation independent of the filters' size. Experimental validation demonstrates that the gain in speed compared to a generic approach can be more than one order of magnitude.

This new technique provides a generic framework for extension of object detection methods, as it relieves all the constraints inherent to sparse and approximate methods. It can in particular be used with any loss, without the need for it to be sparse inducing, and does not require the tuning of any meta-parameter related to sub-sampling or approximate speed-up strategies.

6 Extensions to the original Deformable Part Model

In this chapter we present our contributions and the results of some of our experiments to improve the detection accuracy of a particular linear object detector, namely the Deformable Part Model (DPM) of Felzenszwalb et al. *We investigated several approaches, described in the following sections. After a formal introduction of the standard DPM in § 6.3, we describe the results of our experiments with additional features in § 6.4, our extension to increase the deformability of the model by allowing parts to individually change scale efficiently in § 6.5, and a new model looking jointly at part appearances so as to enforce their consistency in § 6.6. Content presented in this chapter is partly based on the following publication (Dubout and Fleuret, 2013b):*

> C. Dubout and F. Fleuret. Deformable Part Models with Individual Part Scaling. In *British Machine Vision Conference*, 2013.

6.1 Introduction

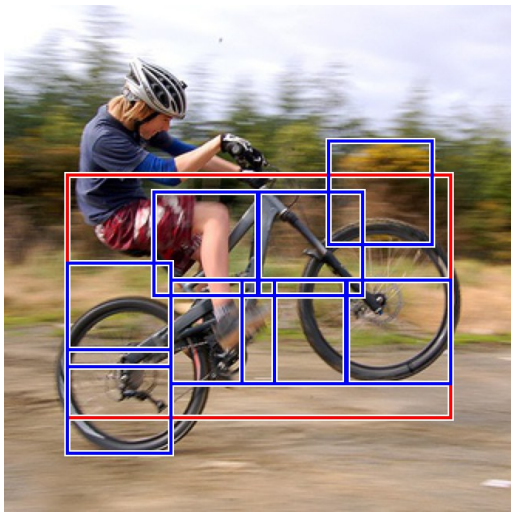
The Deformable Part Model (DPM) of (Felzenszwalb et al., 2010b) and its many variants, some of which are presented in § 6.2, are considered one of the current state-of-the-art object detection methods. Indeed they are the winners of many past Pascal VOC detection challenges (Everingham et al., 2007, 2008, 2009, 2010, 2011), and are the current top-performer on many other detection tasks, *e.g.* pedestrian detection (Felzenszwalb et al., 2010b), bird recognition (Welinder et al., 2010), face detection and feature localization (Zhu and Ramanan, 2012), or articulated pose recognition (Yang and Ramanan, 2011). DPMs are evaluated at a number of positions and scales in an image, predicting each time a discriminative score related to the presence or absence of the object to detect. These scores are computed by taking the sum of the inner products between the model's filters and the corresponding sub-windows of the image, placing each filter at an *optimal* image location. The strength of DPMs resides in their ability to represent an exponential number of templates by letting the part filters float around their reference locations (see figure 6.1 for an illustration), and in finding the optimal part configuration at every possible root position efficiently using a generalized distance transform (Felzenszwalb and Huttenlocher, 2004).

The bulk of their computational cost comes from the numerous convolutions they need to do between every feature pyramid levels and every part filters, each followed by a distance transform. Both can be computed in time linear with the area of the pyramid levels, as we saw in § 4.3.2 and take roughly the same amount of time, as we saw in § 4.5.

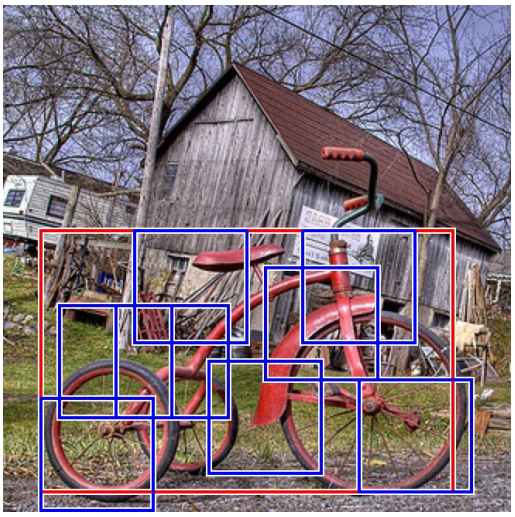
6.2 Related works

Many recent works build upon the original DPM of Felzenszwalb *et al.* and try to improve its detection performance. For example, (Zhang et al., 2011) augments the HOG features usually used with Local Binary Pattern (LBP) ones, in order to be sensitive to not only edges but also textures, which results in a 10% gain in average relative accuracy. Trying to simplify the original star-based part model, (Zhu et al., 2010) represents objects by a mixture of hierarchical tree models organized on a 2D grid, where the nodes represent object parts, and solves the non-convex optimization problem using the Concave-Convex Computational Procedure (CCCP) (Yuille and Rangarajan, 2002). Arguing that the most important component of DPMs is the mixture one, (Divvala et al., 2012) proposes to improve their initialization by switching from aspect-ratio to appearance clustering, and reports that a mixture of monolithic models clustered by appearance can compete with DPMs.

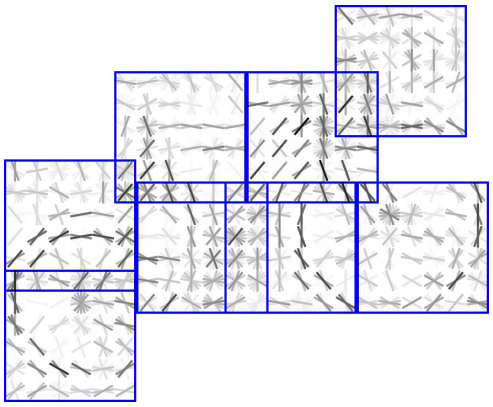
Another line of work (Pepik et al., 2012b,a) aims at bridging the gap between 2D image positions and 3D real-world ones by learning 3D part deformation models. This is accomplished by learning a mixture model where each mixture component deals explicitly with a particular viewpoint, each trained using both real and 3D synthesized images. Even though it enables the model to map 2D part locations to 3D ones, the authors did not attempt to move the parts



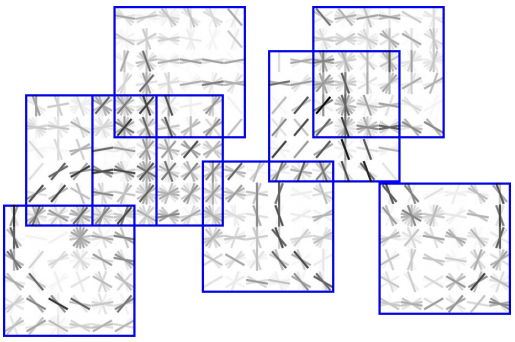
(a) Detection.



(b) Detection.



(c) Part filters.



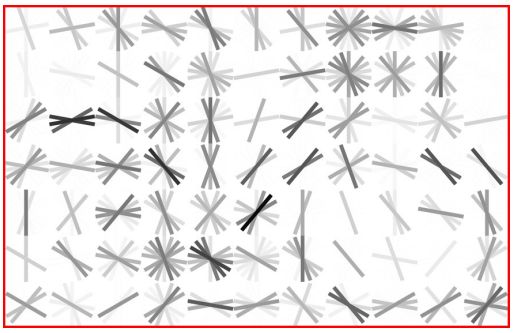
(d) Part filters.

Figure 6.1 – Images (a), (b) show two examples of detections on the Pascal VOC challenge 2007 (Everingham et al., 2007) test set. Images (c), (d) represent the corresponding part filters, which are the same in both images but positioned differently so as to better match the object.

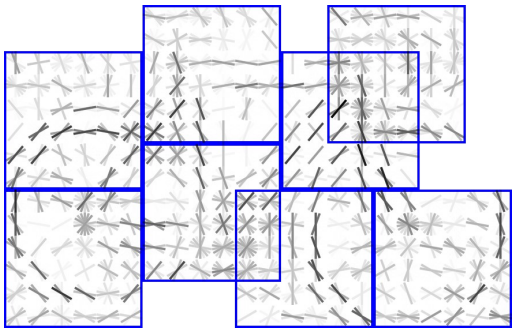
across scales, thus making their work completely orthogonal to the one we present in § 6.5.

6.3 Standard Deformable Part Models

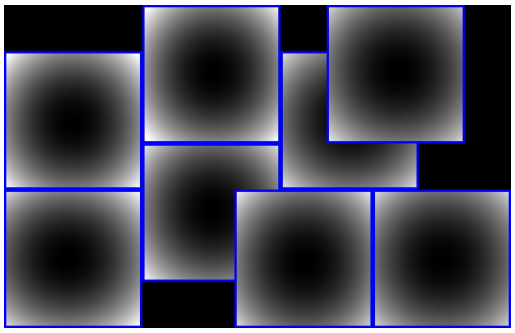
Standard DPMs comprise a root and several parts, all detected independently by a linear filter, and organized in a hierarchical structure specifying the cost of placing the center of a part at different locations relative to the root (see figure 6.2). In the rest of this chapter we restrict ourselves without loss of generality to star structures, for ease of notation. We also ignore the additional complexity introduced by the mixture over the models, usually used to deal with severe changes of appearance, since it is orthogonal to the methods presented here.



(a) Coarse root filter.



(b) Higher resolution part filters.



(c) Spatial model.

Figure 6.2 – A DPM is defined by a coarse root filter (a), several higher resolution part filters (b) and a spatial model for the location of each part relative to the root (c). The filters specify weights for HOG features. Their visualization show the positive weights at different orientations. The visualization of the spatial models reflects the "cost" of placing the center of a part at different locations relative to the root.

Let H be a feature pyramid and $\mathbf{p} = (x, y, z)$ specify a 2D position (x, y) in the z^{th} level of the pyramid. Let $\phi(\mathbf{p})$ denote the vector obtained by concatenating the feature vectors in the sub-window of H centered at $\mathbf{p}$, of dimensions always clear from the context (the dimensions of the filter it is multiplied with), and $\phi_d(\mathbf{p})$ be the deformation features.

A model for an object with n parts is composed of a root filter $\mathbf{w}_0$ and n pairs $(\mathbf{w}_i, \mathbf{d}_i)$, where $\mathbf{w}_i$ is the filter of the i^{th} part and $\mathbf{d}_i$ is a vector specifying the deformation cost of the part placement.

An object hypothesis $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_n$ specifies the location of the center of each filter in a feature pyramid, where the parts are constrained to move in the same level as the root[1]. The score of a hypothesis is then given by the score of each filter at its respective location, minus a deformation cost that depends on the location of each part with respect to the root position

$$S(\mathbf{p}_0, \ldots, \mathbf{p}_n) = \mathbf{w}_0^\top \phi(\mathbf{p}_0) + \sum_{i=1}^{n} \mathbf{w}_i^\top \phi(\mathbf{p}_i) - \mathbf{d}_i^\top \phi_d(\mathbf{p}_i - \mathbf{p}_0). \tag{6.1}$$

[1] Or at a scale which is an integral multiple of the root scale, *e.g.* twice the root resolution in (Felzenszwalb et al., 2010b), since in this case the root positions are a subset of the part ones. In any case this scale is predetermined.

The deformation features are typically

$$\phi_d(\mathbf{p}) = (1, x, y, x^2, y^2) \tag{6.2}$$

in which case it is possible to find the optimal location of each part $\mathbf{p}_i^\star(\mathbf{p}_0)$ as a function of the root position $\mathbf{p}_0$ efficiently (i.e. in time linear with the total number of locations), using a generalized distance transform (Felzenszwalb and Huttenlocher, 2004)

$$\mathbf{p}_i^\star(\mathbf{p}_0) = \underset{\mathbf{p} \in \mathbb{Z}^2 \times \{z_0\}}{\operatorname{argmax}} \mathbf{w}_i^\top \phi(\mathbf{p}) - \mathbf{d}_i^\top \phi_d(\mathbf{p} - \mathbf{p}_0) \tag{6.3}$$

provided that the part locations are integral and limited to a single scale. The score of a root position setting all the parts at their optimal locations is then

$$S(\mathbf{p}_0) = \mathbf{w}_0^\top \phi(\mathbf{p}_0) + \sum_{i=1}^{n} \mathbf{w}_i^\top \phi\big(\mathbf{p}_i^\star(\mathbf{p}_0)\big) - \mathbf{d}_i^\top \phi_d\big(\mathbf{p}_i^\star(\mathbf{p}_0) - \mathbf{p}_0\big). \tag{6.4}$$

6.4 Additional features

We ran experiments using new features in addition to the Histogram of Oriented Gradients (HOG) features (Dalal and Triggs, 2005), modified as in (Felzenszwalb et al., 2010b): histograms of uniform Local Binary Patterns (uLBP), a widely used texture descriptor, and our own color histograms. They are both similar to HOG in the sense that they are local histograms computed over the pixels of each cell of a dense grid, but instead of being histograms of the gradient orientation weighted by the gradient magnitude, they are respectively histograms of the LBP binary code or the hue weighted by the saturation.

6.4.1 Histograms of uniform Local Binary Patterns

Local Binary Pattern (LBP) is a simple yet efficient texture descriptor which labels the pixels of an image by thresholding the 3×3 neighborhood of each pixel with the center value and

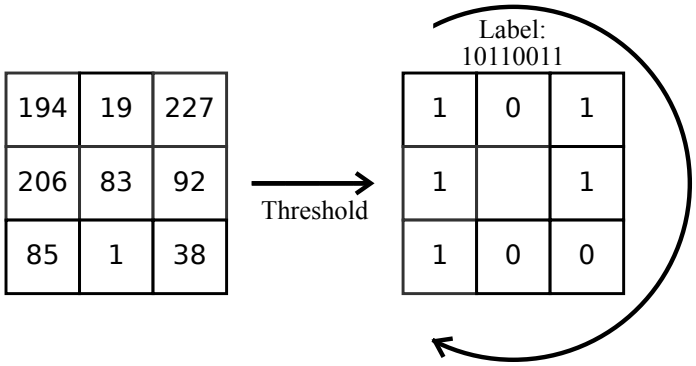
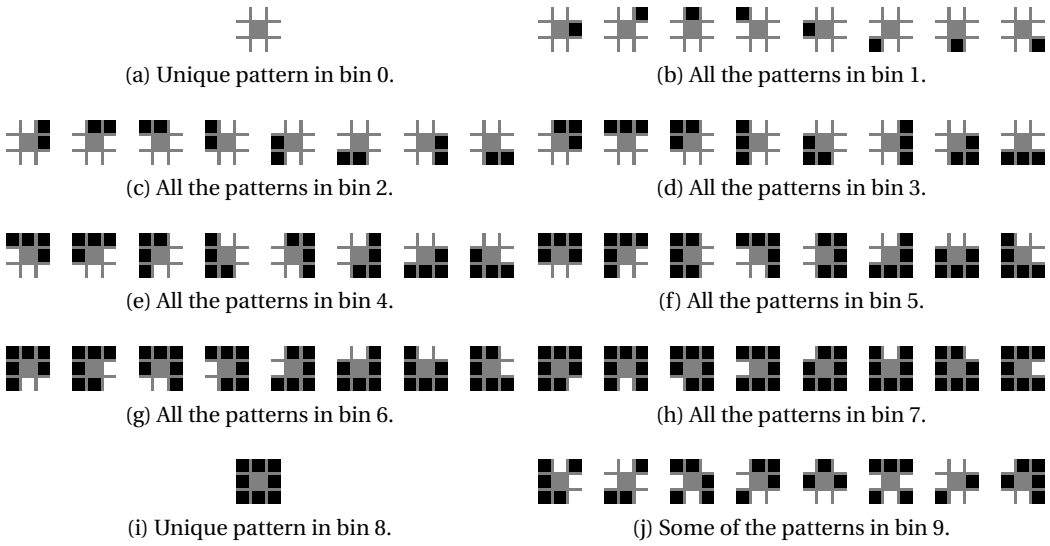


Figure 6.3 – Local Binary Pattern operator using a 3×3 neighborhood.



(a) Unique pattern in bin 0. (b) All the patterns in bin 1.

(c) All the patterns in bin 2. (d) All the patterns in bin 3.

(e) All the patterns in bin 4. (f) All the patterns in bin 5.

(g) All the patterns in bin 6. (h) All the patterns in bin 7.

(i) Unique pattern in bin 8. (j) Some of the patterns in bin 9.

Figure 6.4 – A representation of all the uniform Local Binary Patterns with 0 to 8 bits set (a) to (i), as well as a few of the non uniform patterns (j).

considering the result as a binary number (Ojala et al., 1996) (see figure 6.3). The histogram of these $2^8 = 256$ different labels can then be used as a texture descriptor. In order to reduce the number of labels, we use a circular invariant version of uniform patterns. A local binary pattern is called uniform if it contains at most two bitwise transitions from 0 to 1 or vice versa when the bit pattern is traversed circularly around the center. For example, the patterns 00000000 (0 transitions), 00111000 (2 transitions) and 11110011 (2 transitions) are uniform whereas the patterns 11010001 (4 transitions) and 01010010 (6 transitions) are not. Our histograms contain ten bins: nine bins for the nine possible numbers of bits (0 to 8) set to 1 if the pattern is uniform and one additional bin for all the non uniform patterns (see figure 6.4). There is no notion of magnitude of a pattern, therefore all pixels contribute equally to the local histogram of their corresponding grid cell.

6.4.2 Color histograms

In order to describe the color of a local image patch, we constructed a histogram of the hue weighted by the saturation of the pixels in the Hue, Saturation, and Value (HSV) cylindrical-coordinate representation of the RGB color model. The conversion from RGB to HSV coordi-

(a) Raw pixels. (b) Color histograms.

Figure 6.5 – Representation of the color histograms on a sample image.

nate system is governed by the following equations

$$M = \max(R, G, B) \tag{6.5}$$

$$m = \min(R, G, B) \tag{6.6}$$

$$H = \begin{cases} \frac{G-B}{M-m} \mod 6, & \text{if } M = R \\ \frac{B-R}{M-m} + 2, & \text{if } M = G \\ \frac{R-G}{M-m} + 4, & \text{if } M = B \end{cases} \tag{6.7}$$

$$S = \frac{M-m}{M} \tag{6.8}$$

$$V = M \tag{6.9}$$

where we use the convention $\frac{0}{0} = 0$.

In order for our features to be invariant to linear change of pixel intensities, we do not use the value component V but only the hue H and saturation S. Our features are six bins histograms arranged in the same fashion as HOG on a dense grid, the six bins corresponding to the six integral hues $H = \{0, 1, \ldots 5\}$, which correspond respectively to the red, yellow, green, cyan, blue, and magenta colors. The vote cast by each pixel is proportional to its saturation S, and there is no normalization step as in HOG since the features are already intensity invariant (see figure 6.5).

6.4.3 Experiments

To evaluate the gain in performance provided by the additional features, we re-trained the models of (Felzenszwalb et al., 2011) on all 20 classes of the Pascal VOC 2007 challenge

Table 6.1 – Pascal VOC 2007 challenge Average Precision (area under the Precision/Recall curve) comparison for the models of (Felzenszwalb et al., 2011) re-trained using only HOG features or a combination of HOG, uLBP, and color.

	aero	**bike**	**bird**	**boat**	**bottle**	**bus**	**car**	**cat**	**chair**	**cow**	**table**
HOG (AP)	27.1	59.0	3.7	10.8	25.6	51.1	56.3	14.4	20.6	23.4	20.2
All feat. (AP)	28.8	61.0	4.6	15.7	25.7	49.5	54.7	18.1	20.5	23.6	24.4
Rel. gain (%)	6.5	3.4	23.9	45.2	0.6	-3.0	-2.8	25.6	-0.7	0.9	21.0

	dog	**horse**	**mbike**	**person**	**plant**	**sheep**	**sofa**	**train**	**tv**	**mean**
HOG (AP)	3.3	57.9	50.3	40.3	9.3	16.5	33.9	45.2	39.9	**30.4**
All feat. (AP)	6.2	60.5	51.7	42.5	16.1	22.0	33.0	47.0	44.6	**32.5**
Rel. gain (%)	89.7	4.4	2.7	5.6	73.0	33.3	-2.8	4.1	11.9	**17.1**

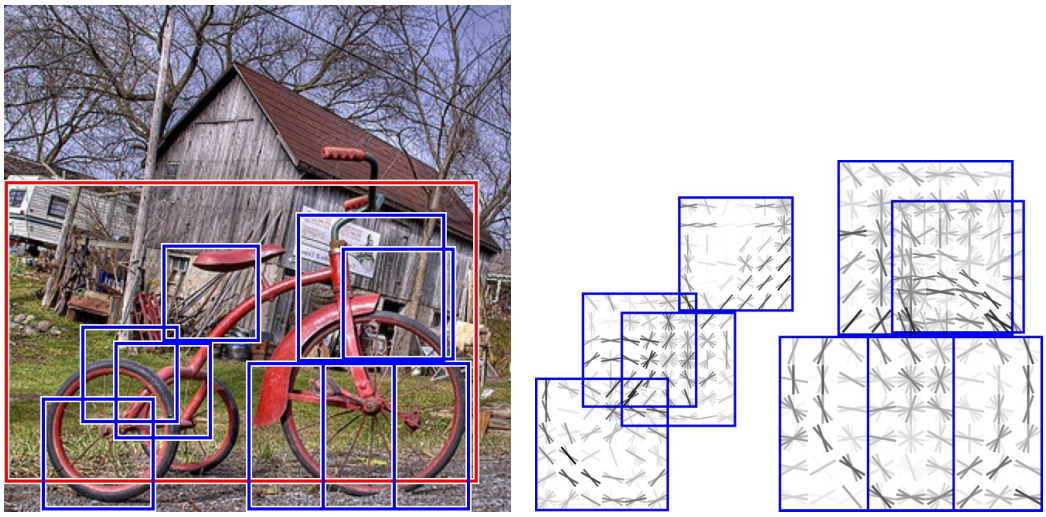
(Everingham et al., 2007) for the same number of iterations using either only the original HOG features or the combination of all three kinds of features.

The performances of the models in both scenarios are displayed in table 6.1. The average precision of the models making use of the additional features improves for 16 of the 20 classes, increasing on average by more than 17%.

6.5 Independent part scaling

Standard DPMs restrict the parts to move at a fixed predetermined scale relative to that of the root of the models (typically at twice the resolution), a limitation imposed by the 2D distance transform efficiently finding the optimal locations of the parts as explained in § 6.5.2. Our extension removes this limitation without increasing the number of convolutions or distance transforms, by sacrificing the guarantee of the optimality of the part placements. Its cost was empirically found to be similar to that of a second distance transform, and it is easy to integrate into existing detection systems, the models staying the same except for the additional 3D components to the deformation cost vectors. Allowing parts to move in 3D increases the expressivity of the models, allowing them to compensate for a wider class of deformations (see 6.6), and might approximate an increase in the scanning resolution. As the number of parameters remains (nearly) constant, overfitting is not a problem.

We introduce our 3D model in § 6.5.1, and our approximation to the generalized distance transform in § 6.5.2. In § 6.5.3 we present results showing an average relative accuracy improvement of 15% compared to standard models, and show that our approximation does not lead to any significant loss of performance by comparing against an exact baseline searching for the optimal part locations exhaustively.



(a) Detection. (b) Part filters.

Figure 6.6 – (a) one of the detections of figure 6.1 this time using a 3D model; (b) the corresponding part filters. Note how the part change scale to better match the object.

6.5.1 Extension to 3D

In our algorithm, we allow the parts to move freely across scales, and extend the deformation features of (6.2) to include the z component of the disparity between root and parts positions

$$\bar{\phi}_d(\mathbf{p}) = (1,\, x,\, y,\, z,\, x^2,\, y^2,\, z^2). \tag{6.10}$$

An illustration of the consequences of allowing parts to move across scales is available in figure 6.7. Ideally, one would like to now find the optimal location of each part in this way

$$\tilde{\mathbf{p}}_i^\star = \underset{\mathbf{p}\in\mathbb{Z}^3}{\operatorname{argmax}} \mathbf{w}_i^\mathsf{T} \phi(\mathbf{p}) - \mathbf{d}_i^\mathsf{T} \bar{\phi}_d\big(\mathbf{p} - \mathbf{p}_0(z_i)\big) \tag{6.11}$$

where $\mathbf{p}_0(z_i) = \big(\lambda^{z_i - z_0}\, x_0,\, \lambda^{z_i - z_0}\, y_0,\, z_0\big)$ are the coordinates of the root position in the i-th part's level z_i, λ being the scaling factor between two successive levels of the feature pyramid. The x and y coordinates of the root position need to be rescaled since they are defined in a different level than the part's coordinates, each level z of the feature pyramid storing features extracted from the image scaled by a factor λ^z. We compare the root and part locations in the part's level, but the particular level at which they are compared does not matter, since each deformation cost $\mathbf{d}_i$ can be scaled accordingly during training.

6.5.2 Approximation to the generalized distance transform

Unfortunately, $\mathbf{p}_0(z_i)$ is likely to be non-integral, and the generalized distance transform thus cannot be used directly anymore. Another issue is that since pyramid levels are of varying sizes, one cannot extend the distance transform to work across levels in the same way as it works

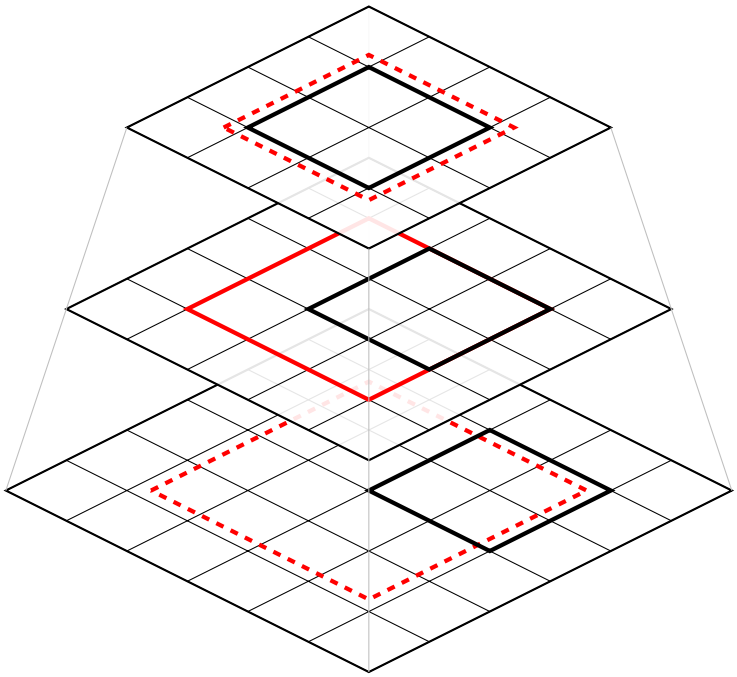


Figure 6.7 – In the middle level of this illustration of a feature pyramid and drawn in solid red is the outline of a root. In the levels above and below and drawn in dashed red are the outlines of the same root scaled to correspond to the same rectangle in the image. In black is the outline of a part deforming across scales. The size of the part is always the size of its filter, here 2×2 HOG cells, which means that it becomes bigger relative to the root in the top level and smaller in the bottom one.

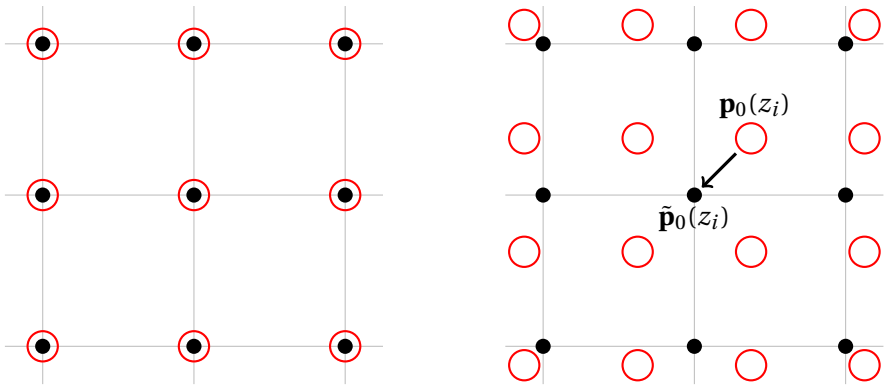
across 2D locations. In order to cope with the first issue, we approximate the root position at the scale of the i-th part by the closest integer one

$$\tilde{\mathbf{p}}_0(z_i) = \underset{\mathbf{p} \in \mathbb{Z}^2 \times \{z_i\}}{\operatorname{argmin}} \left|\left|\mathbf{p} - \mathbf{p}_0(z_i)\right|\right|. \tag{6.12}$$

Using this approximate root position, we can now again use the generalized distance transform in order to find the optimal part location for the approximate root position

$$\tilde{\mathbf{p}}_i^{\star}(\mathbf{p}_0) = \underset{\mathbf{p} \in \mathbb{Z}^3}{\operatorname{argmax}} \mathbf{w}_i^{\mathsf{T}} \phi(\mathbf{p}) - \mathbf{d}_i^{\mathsf{T}} \bar{\phi}_d\left(\mathbf{p} - \tilde{\mathbf{p}}_0(z_i)\right). \tag{6.13}$$

We expect this location to coincide most of the time with the optimal one for the real root position, the difference between the real and the approximate one being at most 0.5 along the x and y axes. However, the optimal score returned by the transform will generally not match the score of any real root and part configuration, and might even be higher than the true optimal one, so we recompute it in constant time using this time the real root position



(a) In standard models. (b) With our 3D model.

Figure 6.8 – Lattices of part locations (in black) in a particular pyramid level. The red circles indicate root positions. In (a) the part and root positions are at the same scale, as is always the case with standard models. In (b) there is a mismatch between the scales of the two, and we show how we approximate a root position $\mathbf{p}_0(z_i)$ by rounding it to the closest integral position $\tilde{\mathbf{p}}_0(z_i)$ when looking for its optimal part placement.

$\mathbf{p}_0(z_i)$ in order to obtain a lower bound on the true optimal score

$$\tilde{S}(\mathbf{p}_0) = \mathbf{w}_0^\top \phi(\mathbf{p}_0) + \sum_{i=1}^{n} \mathbf{w}_i^\top \phi\big(\tilde{\mathbf{p}}_i^\star(\mathbf{p}_0)\big) - \mathbf{d}_i^\top \bar{\phi}_d\big(\tilde{\mathbf{p}}_i^\star(\mathbf{p}_0) - \mathbf{p}_0(z_i)\big)\,. \tag{6.14}$$

The second issue was that the generalized distance transform cannot be extended to work across pyramid levels. Since we can generally expect the number of scales in which parts will deform to be small (much smaller than the number of possible 2D locations at each scale), we brute-force search the optimal level z_i of each part, and for each level use an efficient 2D distance transform. Brute-force searching the optimal level also enables the use of costs other than the quadratic one of (6.10), as long as they remain separable in all dimensions. The results of the transform of (6.13) can be reused for each root level with common part levels, such that by precomputing them for every part level in advance, the total number of transforms is reduced from being quadratic to linear in the total number of pyramid levels.

6.5.3 Experiments

To evaluate our approach to increase the expressivity of DPMs by allowing parts to also move across scales, we trained a mixture of 6 models on all 20 classes of the Pascal VOC 2007 challenge (Everingham et al., 2007). We compare both against standard 2D models as well as "exact" 3D ones, searching exhaustively for the optimal part placement instead of using our approximation of § 6.5.2.

Our DPM implementation is publicly available (Dubout and Fleuret, 2012b) and uses the same modified Histogram of Oriented Gradients (HOG) features (Dalal and Triggs, 2005) and

Table 6.2 – Pascal VOC 2007 challenge Average Precision (area under the Precision/Recall curve) comparison for the models of (Felzenszwalb et al., 2011) as well as our 2D and 3D models. What we call relative gain is the improvement of 3D models over 2D ones.

	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table
voc-release4 (AP)	28.9	60.2	1.7	8.3	20.6	53.5	51.3	6.9	18.7	20.1	13.8
2D DPM (AP)	29.3	58.3	3.6	10.2	23.6	55.6	52.8	9.8	19.1	21.8	22.2
3D DPM (AP)	32.0	60.1	5.1	11.2	27.8	57.3	51.0	19.6	20.5	25.1	23.5
Rel. gain (%)	9.1	3.2	41.2	10.1	17.6	2.9	-3.5	99.7	7.5	15.1	5.7

	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mean
voc-release4 (AP)	3.3	54.5	47.6	38.8	5.8	14.3	28.1	37.3	39.0	**27.6**
2D (AP)	4.3	58.4	46.6	38.6	8.2	17.4	27.7	42.0	41.6	**29.6**
3D (AP)	5.3	60.5	47.0	39.0	10.2	22.1	30.6	42.6	43.1	**31.7**
Rel. gain (%)	23.7	3.5	0.9	1.0	24.5	26.6	10.4	1.5	3.7	**15.2**

the same initialization of the parts locations, sizes, deformation cost, and left/right pose assignments as in (Felzenszwalb et al., 2010b, 2011),. It similarly initializes the parts at twice the resolution of the root (one octave below), and we configured it to always compute 5 scales per octave in the feature pyramid. The only additional parameter relative to the initialization of the 3D models that we needed to specify was the initial deformation cost of the parts, corresponding to the z-coordinate of each vector $\mathbf{d}_i$. We set their linear components to 0 and their quadratic one to 0.01, such that the initial dispersion of parts across scales was approximately centered and of standard deviation 1 level.

While brute-force searching the optimal scale of each part, during both training and testing, we restricted the search to a 7 levels window (± 3 levels) centered on the level one octave below the root, which corresponds to $z_i \in [z_0 - 5 - 3, \ldots, z_0 - 5 + 3]$, meaning that we allowed parts to grow or shrink at most by a factor of $2^{\frac{3}{5}} \approx 1.5$ compared to their reference size. This setting proved sufficient for parts to fully exploit their additional freedom along the z-axis given our initialization of the deformation cost. Since there is some randomness involved in the initialization of the models, we always initialized the seed of the random number generator to the same value while training 2D and 3D models.

To demonstrate the performance of our implementation, we also evaluated the models included in (Felzenszwalb et al., 2011), which achieve close to state-of-the-art detection results, using the same evaluation parameters as our models. These evaluation parameters might not be optimal for those models, and we therefore include their results as a reference point only.

Table 6.3 – Pascal VOC 2007 challenge AP comparison on the first 100 images of each class for the exact as well as our approximation to the generalized distance transform method of § 6.5.2 using our 3D models.

	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table
Brute-force (AP)	50.1	69.0	18.0	20.9	38.8	72.7	59.1	33.0	28.8	46.5	47.3
Approx. DT (AP)	49.9	69.2	17.9	21.5	38.5	72.7	59.1	32.4	28.7	46.3	47.3

	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mean
Brute-force (AP)	18.7	77.2	60.3	32.9	23.9	37.6	44.3	56.0	64.3	**45.0**
Approx. DT (AP)	18.4	77.2	60.3	33.0	24.6	37.2	44.3	56.0	64.2	**44.9**

6.5.4 Results

The performances of all 3 kinds of models on the Pascal VOC 2007 challenge are displayed in table 6.2. The scoring function of the Pascal VOC development kit computes the average precision score for each model by sampling the Precision/Recall curve in eleven points of recall 0.0, 0.1, 0.2, …, 1.0. In order to increase its precision, which is particularly important for difficult classes obtaining less than 10% AP, we modified it to take into account all points of the PR curve. This modification explains why the scores we obtain on some difficult classes are lower than usually reported, and why some authors obtain scores above 9% AP while correctly detecting only one object on the whole dataset (since the first point is sampled with recall 0.0, as long as the first detection is correct the AP is guaranteed to be at least $\frac{1}{11}$).

The average precision of our 3D models improves over the 2D ones for 19 of the 20 classes, increasing on average by more than 15%. The average time taken by our implementation to detect objects in an image using one of the 2D models was 77 ms. Out of those 77 ms, we measured that 22 were spent computing distance transforms. When using a 3D model, the average total time increased to 99 ms, corresponding to a doubling of the transform time.

Apart from increasing the expressivity of the models, allowing parts to move across scales might also improve performance by simulating a scan of the image at a higher resolution. This may happen because HOG grids in neighboring pyramid levels have always the same step size (typically 8 pixels), but slightly different dimensions. This difference intertwines their positions on the image as in figure 6.8 b), and may make the whole process similar to searching for objects on several slightly misaligned HOG grids. Another explanation is that our model allows the root to go lower in the feature pyramid, as it is not forced to be one octave above the parts anymore, visiting a somewhat higher number of negative examples.

A comparison of our approximate method versus the “exact” one which brute-force searches the optimal location of each part is shown in table 6.3. We evaluated both methods only on the first 100 images of each class because of the prohibitive time taken by the exhaustive search.

These results demonstrate the accuracy of the approximation.

6.6 Joint appearance constraints

A shortcoming of standard DPMs is that they completely ignore joint aspects of appearance, marginalizing it completely over the parts. We believe that making sure that the appearance of a candidate part agrees with the appearance of the root would be useful to weed out invalid part placements, and thus reduce the score of false positives.

We therefore propose to extend the standard part score of equation (6.1) in § 6.3 with a term looking jointly at the appearance of the part and the root together in the following manner

$$S^{\dagger}(\mathbf{p}_0, \ldots, \mathbf{p}_n) = \mathbf{w}_0^{\top}\phi(\mathbf{p}_0) + \sum_{i=1}^{n} \mathbf{w}_i^{\top}\phi(\mathbf{p}_i) - \mathbf{d}_i^{\top}\phi_d(\mathbf{p}_i - \mathbf{p}_0) - \left|\left|\mathbf{A}_i'\phi(\mathbf{p}_0) - \mathbf{A}_i''\phi(\mathbf{p}_i)\right|\right|^2 \quad (6.15)$$

$$= \mathbf{w}_0^{\top}\phi(\mathbf{p}_0) + \sum_{i=1}^{n} \mathbf{w}_i^{\top}\phi(\mathbf{p}_i) - \mathbf{d}_i^{\top}\phi_d(\mathbf{p}_i - \mathbf{p}_0) - \left|\left|\mathbf{A}_i \begin{pmatrix} \phi(\mathbf{p}_0) \\ -\phi(\mathbf{p}_i) \end{pmatrix}\right|\right|^2 \quad (6.16)$$

where $\mathbf{A}_i = \begin{pmatrix}\mathbf{A}_i' & \mathbf{A}_i''\end{pmatrix}$. This formulation has a number of advantages over other joint ones:

First it is simple to understand. The projections matrices $\mathbf{A}_i'$ and $\mathbf{A}_i''$ project respectively the root and part i into a low-dimensional subspace into which it is meaningful to measure similarity using the Euclidean distance. Vectors in that subspace can be interpreted as a soft mixture assignment where the mixture components are the rows of $\mathbf{A}_i$.

Second, it is computationally much more manageable than jointly looking at all the features together, as it still decomposes into a sum over parts, and as the projections (the rows of $\mathbf{A}_i'$ and $\mathbf{A}_i''$) are simple linear filters similar to the $\mathbf{w}_i$s, and thus can be accelerated by the method of § 4.3.2. We do expect that ignoring part – part relations will lead to a loss in discriminative power, but we expect it to be mild since the root appearance already contains the part appearance, albeit at a lower resolution.

Third since distances are non-negative the original scores $S(\mathbf{p}_0)$ always upper bound their joint variant $S^{\dagger}(\mathbf{p}_0)$, allowing one to use them in a cascade or a branch-and-bound algorithm.

A drawback inherent to any method looking jointly at part appearances is that the efficient distance transform of (Felzenszwalb and Huttenlocher, 2004) can no longer be applied, and one as to fall back to exhaustive search. Fortunately we observed empirically that one does not need to look into a large region of the space for the optimal location of each part (the maximum over $\mathbf{p}_i$ in (6.4) and (6.15)) but can restrict himself into a small region centered on the reference anchor point.

We propose two variants of the above formulation, differing in their divergence from the original DPM formulation and their computational complexity.

6.6.1 Post-scoring (DPM†)

The first variant is to use the extended score of (6.15) but to keep the original part configuration of $S(\mathbf{p}_0)$ (6.4)

$$S^{\dagger}(\mathbf{p}_0) = S^{\dagger}(\mathbf{p}_0, \mathbf{p}_1^{\star}(\mathbf{p}_0), \ldots, \mathbf{p}_n^{\star}(\mathbf{p}_0)). \tag{6.17}$$

It has the advantage of being simple and efficient, and is trivial to cascade since it can be done as a post-processing step.

6.6.2 Joint-scoring (DPM‡)

The second variant optimizes the parts' optimal locations using the new score of (6.15)

$$S^{\ddagger}(\mathbf{p}_0) = \underset{\mathbf{p}_1, \ldots, \mathbf{p}_n}{\operatorname{argmax}} S^{\dagger}(\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_n). \tag{6.18}$$

It is more powerful than the first variant, as it looks jointly at the root and part appearances when looking for the optimal part placement, but is also much more expensive since a distance transform cannot be used anymore.

6.6.3 Learning

A drawback of the previous formulation is that the hypothesis scores are no longer fully linear in their parameters, leading to a more complex learning problem (non-convex), sensitive to the initialization of the $\mathbf{A}_i$s.

We apply the method of discriminant embedding of (Hua et al., 2007) to initialize the projections, which was originally proposed to discriminate between matching and non-matching image patches in order to train a compact local image descriptor. We modified its objective function to take pairs of root – part appearance vectors instead of pairs of image patches as follows

$$J(\mathbf{a}_i) = \frac{\sum_{j, y_j=-1} \left(\mathbf{a}_i^{\mathsf{T}} \begin{pmatrix} \phi(\mathbf{p}_{0,j}) \\ -\phi(\mathbf{p}_{i,j}) \end{pmatrix} \right)^2}{\sum_{j, y_j=+1} \left(\mathbf{a}_i^{\mathsf{T}} \begin{pmatrix} \phi(\mathbf{p}_{0,j}) \\ -\phi(\mathbf{p}_{i,j}) \end{pmatrix} \right)^2} \tag{6.19}$$

where $\mathbf{a}_i$ stands for an arbitrary row of $\mathbf{A}_i$, y_j is the label and $\mathbf{p_{i,j}}$ is the location of part i for example j. The solution of this equation is the largest eigenvector of the system

$$\mathbf{Na} = \lambda \mathbf{Pa} \tag{6.20}$$

where $\mathbf{N}$ and $\mathbf{P}$ are the covariance matrices respectively of the negative and positive examples.

Since we want to learn more than one projection we keep the k eigenvectors corresponding to the k largest eigenvalues.

6.6.4 Experiments

We kept all the original parameters of the model (all the filters $\mathbf{w}_i$ and deformation cost $\mathbf{d}_i$) fixed and attempted to learn only the $\mathbf{A}_i$s on the Pascal VOC challenge 2007 (Everingham et al., 2007). We were unable to significantly increase the detection accuracy over the original model, probably because of overfitting. As the performance of the original model is already close to perfect on the training set, and since the number of positive examples (a few hundreds) is small, it is hard to learn meaningful projections. We leave as future work the task of learning the projections either on a larger dataset or with a better regularization strategy than the Euclidean norm that we employed.

6.7 Conclusion

We proposed three extensions to the original Deformable Part Model of (Felzenszwalb et al., 2010b) all taking advantages of the acceleration strategies described in the previous two chapters. The first one, the addition of LBP and color features, increases on average the detection accuracy of the models by 17%, the models remaining exactly the same. The second one, allowing parts to independently change scale, increases on average the detection accuracy of the models by 15% for a moderate augmentation of the total computational cost, the number of convolutions and distance transforms remaining constant. Despite relying on an approximation to the generalized distance transform, it obtains scores virtually equal to an exact brute-force search. Even though we were unable to empirically demonstrate the advantage of our third extension, modeling the appearance of the parts jointly with the one of the root, we are confident about the potential of our approach, as real-world object parts are rarely independent. We believe its non-performance to be due to our inability to train it on the Pascal dataset.

7 Summary and Future Directions

This chapter summarizes the main results and contributions presented in this thesis, and sketches possible future directions of research.

7.1 Discussion

In this thesis we studied the problem of learning and detecting objects in high dimensional feature space. In the first part, we showed that integrating information about the division of the feature space into homogeneous subsets can reduce the training time significantly. In particular, the Tasting algorithm of § 3.4 is extremely straight-forward and avoids the need for setting multiple parameters, such as the trade-off between exploration and exploitation. In practice, the only parameter to set is the number of features R sampled initially to estimate the expected loss reduction during training. Tasting relies on the ability to estimate the loss reduction given any weighting of the training samples. We have chosen to use an empirical model, that is to store actual responses over samples, instead of fitting an analytical density model. It may be possible to choose the later strategy, and summarize the information provided by feature responses for instance with a Gaussian model. However, it is not clear how such a model could lead to a proper estimate of the distribution of the loss reduction when the Boosting weights are strongly unbalanced.

We also improved Boosting by modeling the statistical behavior of the weak learners' edges, explicitly with the Maximum Adaptive Sampling (M.A.S.) algorithm and implicitly with the Laminating algorithm. This allowed us to maximize the loss reduction of each weak learner under strict control of the computational cost. Both algorithms are (nearly) parameter-less and perform well on real-world pattern recognition tasks, especially for a small number of iterations, as detailed in § 3.6.5.

In the second part, we showed that it is possible to accelerate without approximation the speed of a large class of linear object detectors, and proposed several enhancements to one of them: the Deformable Part Model (DPM) of Felzenszwalb *et al.* The idea motivating our work

is that the Fourier transform is linear, enabling one to do the addition of the convolutions across the K feature planes in the frequency domain, and be left in the end with only one inverse Fourier transform to do, instead of K. To take advantage of this, we proposed several additional implementation strategies, ensuring maximum efficiency without requiring huge memory space and/or bandwidth, and thus making the whole approach practical. We have also presented a novel method to speed up the training of object detectors based on a linear classifier. Existing implementations of such methods rely on sparsity and sub-sampling of the training examples. Our approach by contrast, is based on a formulation of the gradient computation as a convolution, which allows to leverage the Fourier transform, and makes the overall computation independent of the filters' size. Experimental validation demonstrates that the gain in speed compared to a generic approach can be more than one order of magnitude. That such approaches are possible is not entirely trivial (the reference implementation of Felzenszwalb et al. (2011) contains five different ways to do the convolutions, all at least an order of magnitude slower); nevertheless, the analysis we developed is readily applicable to many other systems. Finally our extensions to the original DPM make full use of our acceleration strategies and increase on average the detection accuracy by 15% for a moderate augmentation of the total computational cost.

7.2 Future Directions

Extensions of the methods proposed in the first part could be investigated along three axes. The first one is to merge the best two methods by adding a Tasting component to the Laminating procedure, in order to bias the sampling towards promising feature subsets. The second is to add a bandit-like component to the methods by adding a variance term related to the lack of samples, and their obsolescence in the Boosting process. This would account for the degrading density estimation when subsets have not been sampled for a while, and induce an exploratory sampling which may be missing in the current algorithms. The third would be to take memory into account, by caching and reusing examples and/or features across multiple Boosting iterations. There is an obvious trade-off between reusing data to save time, and keeping an unbiased estimate of the complete training set, which is not taken into account by current methods.

Regarding the second part, two potential avenues of research come to mind. The first one is to improve the training of our DPM with joint appearance constraints, either by using more training examples or by improving on the initialization and the regularization of the projections. The second is to adapt our accelerated learning framework for linear object detector to Convolutional Neural Networks (CNN), which are usually trained using stochastic gradient descent and whose gradient can be rewritten as a sequence of convolutions.

A Proof of Lemma 1

Since

$$\max_{r:\epsilon_r \geq \epsilon_{q^\star}-\delta} \hat{\epsilon}_r \geq \hat{\epsilon}_{q^\star} \tag{A.1}$$

Defining $B_q = \mathbf{1}_{\{\Delta_q - \Delta_{q^\star} \geq \delta\}}$, we have

$$\begin{aligned}
&\mathrm{P}\left(\left|\left\{q:\ \epsilon_q \leq \epsilon_{q^\star}-\delta,\ \hat{\epsilon}_q \geq \max_{r:\epsilon_r \geq \epsilon_{q^\star}-\delta} \hat{\epsilon}_r\right\}\right| \ \geq\ \frac{Q}{2}\right) \\
\leq &\mathrm{P}\left(\left|\left\{q:\ \epsilon_q \leq \epsilon_{q^\star}-\delta,\ \hat{\epsilon}_q \geq \hat{\epsilon}_{q^\star}\right\}\right| \ \geq\ \frac{Q}{2}\right) && \text{(A.2)} \\
\leq &\mathrm{P}\left(\left|\left\{q:\epsilon_q \leq \epsilon_{q^\star}-\delta, \Delta_q - \Delta_{q^\star} \geq \delta\right\}\right| \geq \frac{Q}{2}\right) && \text{(A.3)} \\
\leq &\mathrm{P}\left(\sum_{q,\epsilon_q \leq \epsilon_{q^\star}-\delta} B_q \geq \frac{Q}{2}\right) && \text{(A.4)} \\
\leq &\mathrm{P}\left(\sum_q B_q \geq \frac{Q}{2}\right) && \text{(A.5)} \\
\leq &\mathrm{P}\left(\frac{2\sum_q B_q}{Q} \geq 1\right) && \text{(A.6)} \\
\leq &\frac{2\,\mathrm{E}[\sum_q B_q]}{Q} && \text{(A.7)} \\
\leq &2\,\mathrm{E}[B_q] && \text{(A.8)} \\
\leq &2\,\mathrm{E}\left[\mathbf{1}_{\{(\Delta_q \geq \frac{\delta}{2}) \cup (\Delta_{q^\star} \leq -\frac{\delta}{2})\}}\right] && \text{(A.9)} \\
\leq &4\exp\left(-\frac{\delta^2 S}{2}\right). && \text{(A.10)}
\end{aligned}$$

■

Equation (A.1) is true since $q^\star$ is among the $\{r : \epsilon_r \geq \epsilon_{q^\star} - \delta\}$ and δ is positive. Equations (A.2) to (A.6) are true since we relax conditions on the event. Equation (A.7) is true since $\mathrm{P}(X \geq 1) \leq \mathrm{E}(X)$ for $X \geq 0$. Equations (A.8) and (A.9) are true analytically, and equation (A.10) follows from Hœffding's inequality.

B Proof of Theorem 1

Defining $\delta_k = \frac{1}{C}\delta\sqrt{\frac{k}{2^{k-1}}}$ where $C = \sum_{k=1}^{\lceil \log_2(Q) \rceil} \sqrt{\frac{k}{2^{k-1}}}$ is a normalization constant such that the δ_k's sum to the original δ, i.e. $\sum_{k=1}^{\lceil \log_2(Q) \rceil} \delta_k = \delta$.

We apply Lemma 1 with constant δ_k for each of the k Laminating iterations, $1 \leq k \leq \lceil \log_2(Q) \rceil$. Since each iteration samples twice as many training examples as the previous one, and the δ_k's sum to the original δ, the probability to end up with a weak learner with an edge below or equal to $\epsilon_{q^*} - \delta$ is upper bounded by

$$
\begin{aligned}
& 4 \sum_{k=1}^{\lceil \log_2(Q) \rceil} \exp\left(-\frac{\delta_k^2 S 2^{k-1}}{2}\right) \\
\leq & 4 \sum_{k=1}^{\infty} \exp\left(-\frac{\delta^2 S k}{2C^2}\right) && \text{(B.1)} \\
\leq & 4 \left(\frac{1}{1 - \exp\left(-\frac{\delta^2 S}{2C^2}\right)} - 1\right) && \text{(B.2)} \\
\leq & 4 \left(\frac{1}{1 - \exp\left(-\frac{\delta^2 S}{69}\right)} - 1\right) && \text{(B.3)}
\end{aligned}
$$

■

Equation (B.1) is true analytically, equation (B.2) follows from the formula for geometric series, and equation (B.3) is true due to the fact that the constant C is upper bounded by $\sqrt{2}$ times the polylogarithm $\mathrm{Li}_{-\frac{1}{2}}\left(\frac{1}{\sqrt{2}}\right) = \sum_{k=1}^{\infty} \sqrt{\frac{k}{2^k}} \approx 4.15$.

Bibliography

P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002.

P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Scientific Computing*, 32(1):48–77, 2003.

F. R. Bach, G. R. G. Lanckriet, and M. I. Jordan. Multiple kernel learning, conic duality, and the SMO algorithm. In *International Conference on Machine Learning*, 2004.

L. Bottou and Y. LeCun. Large scale online learning. In *Neural Information Processing Systems*, 2003.

J. Bradley and R. Schapire. Filterboost: Regression and classification on large datasets. In *Neural Information Processing Systems*, 2007.

R. Busa-Fekete and B. Kegl. Accelerating AdaBoost using UCB. *JMLR W&CP*, 2009.

R. Busa-Fekete and B. Kegl. Fast Boosting using adversarial bandits. In *ICML*, 2010.

H. Cecotti and A. Graeser. Convolutional neural network with embedded fourier transform for eeg classification. In *International Conference on Pattern Recognition*, pages 1–4, 2008.

B. Chazelle. The bottom-left bin-packing heuristic: An efficient implementation. In *IEEE Transactions on Computers*, pages 697–707, 1983.

N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Conference on Computer Vision and Pattern Recognition*, pages 886–893, 2005. URL http://pascal.inrialpes.fr/data/human/.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: a large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition*, 2009.

S. Divvala, A. Efros, and M. Hebert. How important are "deformable parts" in the deformable parts model? In *European Conference on Computer Vision*, pages 31–40, 2012.

P. Dollar, S. Belongie, and P. Perona. The Fastest Pedestrian Detector in the West. In *British Machine Vision Conference*, 2010. URL http://bmvc10.dcs.aber.ac.uk/proc/conference/paper68/index.html.

C. Dubout and F. Fleuret. Tasting families of features for image classification. In *International Conference on Computer Vision*, 2011a.

C. Dubout and F. Fleuret. Boosting with maximum adaptive sampling. In *Neural Information Processing Systems*, 2011b.

C. Dubout and F. Fleuret. Exact acceleration of linear object detectors. In *Computer Vision – ECCV 2012*, volume 7574 of *Lecture Notes in Computer Science*, pages 301–311. Springer Berlin Heidelberg, 2012a. URL http://dx.doi.org/10.1007/978-3-642-33712-3_22.

C. Dubout and F. Fleuret. Exact acceleration of linear object detectors, 2012b. URL http://www.idiap.ch/scientific-research/resources/exact-acceleration-of-linear-object-detectors.

C. Dubout and F. Fleuret. Accelerated training of linear object detectors. In *CVPR 2013 Workshop on Structured Prediction*, 2013a. URL cvpr13ws.is.tue.mpg.de.

C. Dubout and F. Fleuret. Deformable part models with individual part scaling. In *British Machine Vision Conference*, 2013b.

N. Duffield, C. Lund, and M. Thorup. Priority sampling for estimation of arbitrary subset sums. *J. ACM*, 54, December 2007.

G. Escudero, L. Màrquez, and G. Rigau. Boosting applied to word sense disambiguation. *Machine Learning: ECML 2000*, pages 129–141, 2000.

M. Everingham, A. Zisserman, C. K. I. Williams, and L. Van Gool. The PASCAL Visual Object Classes Challenge 2006 (VOC2006) Results, 2006. URL http://www.pascal-network.org/challenges/VOC/voc2006/results.pdf.

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results, 2007. URL http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2008 (VOC2008) Results, 2008. URL http://www.pascal-network.org/challenges/VOC/voc2008/workshop/index.html.

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2009 (VOC2009) Results, 2009. URL http://www.pascal-network.org/challenges/VOC/voc2009/workshop/index.html.

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Results, 2010. URL http://www.pascal-network.org/challenges/VOC/voc2010/workshop/index.html.

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2011 (VOC2011) Results, 2011. URL http://www.pascal-network.org/challenges/VOC/voc2011/workshop/index.html.

R. Fan, K. Chang, C. Hsieh, X. Wang, and C. Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 6 2008.

L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. In *CVPR, Workshop on Generative-Model Based Vision*, 2004. URL http://www.vision.caltech.edu/Image_Datasets/Caltech101/.

P. Felzenszwalb and D. Huttenlocher. Distance transforms of sampled functions. Technical report, Cornell Computing and Information Science, 2004.

P. Felzenszwalb and D. Huttenlocher. Pictorial structures for object recognition. In *International Journal of Computer Vision*, 2005.

P. Felzenszwalb, R. Girshick, and D. McAllester. Cascade object detection with deformable part models. In *Conference on Computer Vision and Pattern Recognition*, pages 2241–2248, 2010a.

P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object Detection with Discriminatively Trained Part-Based Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010b.

P. Felzenszwalb, R. Girshick, and D. McAllester. Discriminatively trained deformable part models, release 4, 2011. URL http://cs.brown.edu/~pff/latent-release4/.

F. Fleuret and D. Geman. Stationary features and cat detection. *Journal of Machine Learning Research*, 9:2549–2578, 2008.

F. Fleuret, T. Li, C. Dubout, E. K. Wampler, S. Yantis, and D. Geman. Comparing machines and humans on a visual categorization test. *Proceedings of the National Academy of Sciences*, 2011.

Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156. Morgan Kaufmann, 1996.

J. H. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of Boosting. *Annal of Statistics*, 28:337–407, 2000.

M. Frigo and S. Johnson. The design and implementation of FFTW3. In *Proceedings of the IEEE*, volume 93 (2), pages 216–231, 2005.

P. Gehler and S. Nowozin. On feature combination for multiclass object classification. In *International Conference on Computer Vision*, 2009.

G. Guennebaud, B. Jacob, et al. Eigen v3, 2010. URL http://eigen.tuxfamily.org.

C. Hsieh, K. Chang, C. Lin, S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear svm. In *International Conference on Machine Learning*, pages 408–415, 2008.

G. Hua, M. Brown, and S. A. J. Winder. Discriminant embedding for local image descriptors. In *International Conference on Computer Vision*, pages 1–8, 2007.

T. Joachims. Training linear svms in linear time. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 217–226, 2006.

Z. Kalal, J. Matas, and K. Mikolajczyk. Weighted sampling for large-scale Boosting. *British machine vision conference*, 2008.

I. Kokkinos. Rapid deformable object detection using dual tree branch and bound. In *Advances in Neural Information Processing Systems*, 2011.

I. Kokkinos. Bounding part scores for rapid detection with deformable part models, 2012.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Computer Science University of Toronto, 2009. URL http://www.cs.toronto.edu/~kriz/cifar.html.

C. Lampert, M. Blaschko, and T. Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

C. H. Lampert, M. B. Blaschko, and T. Hofmann. Efficient subwindow search: A branch and bound framework for object localization. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 31, pages 2129–2142, 2009.

G. R. G. Lanckriet, N. Cristianini, P. Bartlett, L. El Ghaoui, and M. I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5, 2004.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998a.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86(11), pages 2278–2324, 1998b. http://yann.lecun.com/exdb/mnist/.

S. Maji and J. Malik. Object detection using a max-margin hough transform. In *Conference on Computer Vision and Pattern Recognition*, pages 1038–1045, 2009.

M. E. Nilsback and A. Zisserman. A visual vocabulary for flower classification. In *International Conference on Computer Vision*, pages 1447—1454, 2006.

T. Ojala, M. Pietikainen, and D. Harwood. A comparative study of texture measures with classification based on feature distributions. In *Pattern Recognition 19*, volume 3, pages 51–59, 1996.

A. Opelt, A. Pinz, M. Fussenegger, and P. Auer. Generic object recognition with Boosting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28:416–431, 2006.

B. Pepik, P. Gehler, M. Stark, and B. Schiele. 3D^{2}PM - 3D deformable part models. In *European Conference on Computer Vision*, 2012a.

B. Pepik, M. Stark, P. Gehler, and B. Schiele. Teaching 3d geometry to deformable part models. In *Conference on Computer Vision and Pattern Recognition*, 2012b.

R. Perko and A. Leonardis. Context driven focus of attention for object detection. In *Attention in Cognitive Systems. Theories and Systems from an Interdisciplinary Viewpoint (WAPCV 2007)*, volume 4840, chapter 14, pages 216–233. Springer LNAI, 2007. URL http://vicos.fri.uni-lj.si/data/publications/perko07context_driven.pdf.

H. Pirsiavash and D. Ramanan. Steerable part models. In *Conference on Computer Vision and Pattern Recognition*, 2012.

J. Platt. Advances in kernel methods. In *Fast training of support vector machines using sequential minimal optimization*, pages 185–208, 1999.

R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine learning*, 37(3):297–336, 1999.

R. E. Schapire, Y. Freund, P. L. Bartlett, and W. S. Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26(5):1651–1686, 1998.

J. Schatzman. Accuracy of the discrete fourier transform and the fast fourier transform. *SIAM Journal on Scientific Computing*, 17(5):1150–1166, 1996.

S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *International Conference on Machine Learning*, pages 807–814, 2007.

V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

P. Viola and M. Jones. Robust real-time object detection. In *International Journal of Computer Vision*, 2001.

P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

R. Wijnhoven and P. de With. Fast training of object detection using stochastic gradient descent. In *International Conference on Pattern Recognition*, pages 424–427, 2010.

Y. Yang and D. Ramanan. Articulated pose estimation using flexible mixtures of parts. In *Conference on Computer Vision and Pattern Recognition*, 2011.

A. Yuille and A. Rangarajan. The concave-convex procedure (cccp. In *Neural Information Processing Systems*, 2002.

C. Zhang and P. Viola. Multiple-instance pruning for learning efficient cascade detectors. In *Neural Information Processing Systems*, 2007.

J. Zhang, K. Huang, Y. Yu, and T. Tan. Boosted local structured hog-lbp for object localization. In *Conference on Computer Vision and Pattern Recognition*, pages 1393–1400, 2011.

L. Zhu, Y. Chen, A. Yuille, and W. Freeman. Latent hierarchical structural learning for object detection. In *Conference on Computer Vision and Pattern Recognition*, pages 1062–1069, 2010.

X. Zhu and D. Ramanan. Face detection, pose estimation, and landmark localization in the wild. In *Conference on Computer Vision and Pattern Recognition*, pages 2879–2886, 2012.

Charles Dubout

CONTACT INFORMATION

Rue de Crissier 12
1020 Renens
Switzerland
cdubout@gmail.com

Cellphone: (+41) 79 697 96 89
Date of Birth: September 8, 1985
Citizenships: Swiss, French
www.idiap.ch/~cdubout

CURRENT POSITION

Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland

Ph.D., Computer Science — **Fall 2009 to Winter 2013**

- Working at the **Idiap Research Institute** with **Dr. François Fleuret**
- Thesis title is "Object Classification and Detection in High Dimensional Feature Space", involves both Machine Learning and Computer Vision
- Graduated with a GPA of 5.8 on a scale of 6.0

EDUCATION

Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland

M.S., Computer Science — **Fall 2006 to Summer 2008**

- Graduated with a GPA of 5.6 on a scale of 6.0

B.S., Computer Science — **Fall 2003 to Summer 2006**

- Graduated with a GPA of 5.3 on a scale of 6.0

PROFESSIONAL EXPERIENCE

Qualcomm Corporate R&D, San Diego, United States

Interim Engineering Intern — **Summer 2013**

- Mobile vision

NEC Corporation, Tokyo, Japan

System Engineer — **Winter 2008 to Summer 2009**

- Continuation of my *M.S. Thesis*

M.S. Thesis — **Spring and Summer 2008**

- Face pictures enhancement based on *CG* rendering approaches

AgieCharmilles, Meyrin, Switzerland

Computer Technician — **Summer 2006 and 2007**

- Development of a standalone maintenance tool

Technical Skills

Machine Learning and Computer Vision

- Discriminative Machine Learning (SVM, Boosting, and Neural Nets), Deformable Part Models, Feature Detection and Description, Multiple View Geometry

Programming

- C/C++ (expert), Matlab (advanced), x86 assembly, Java, Perl, Python, UNIX shell

Application design

- Author of the **Fast Fourier Linear Detector** software (the fastest public implementation of Deformable Part Models), as well as an **MSER detector**, a **SIFT descriptor**, implementations of the **L-BFGS** and the **FFT** algorithms, *etc.*, all implemented in C++ and distributed online under the GNU GPL

Selected Publications

2013

- **Adaptive Sampling for Large Scale Boosting**, Charles Dubout and François Fleuret, in *Journal of Machine Learning Research* (JMLR)
- **Deformable Part Models with Individual Part Scaling**, Charles Dubout and François Fleuret, in *British Machine Vision Conference* (BMVC)
  Acceptance rate: 30%
- **Accelerated Training of Linear Object Detectors**, Charles Dubout and Fran-çois Fleuret, in *IEEE Conference on Computer Vision and Pattern Recognition, Structured Prediction workshop* (CVPR workshop)

2012

- **Exact Acceleration of Linear Object Detectors**, Charles Dubout and François Fleuret, in *Proceedings of the European Conference on Computer Vision* (ECCV)
  Oral presentation, acceptance rate: 2.8%
- Co-inventor of the US patent application US13/624375: **Object detection method, object detector and object detection computer program**

2011

- **Boosting with Maximum Adaptive Sampling**, Charles Dubout and François Fleuret, in *Proceedings of the Neural Information Processing Systems Conference* (NIPS)
  Acceptance rate: 22%, Idiap Best Student Paper Award
- **Comparing machines and humans on a visual categorization test**, François Fleuret, Ting Li, Charles Dubout, Emma K. Wampler, Steven Yantis and Donal Geman, in *Proceedings of the National Academy of Sciences* (PNAS)
- **Tasting Families of Features for Image Classification**, Charles Dubout and François Fleuret, in *International Conference on Computer Vision* (ICCV)
  Acceptance rate: 24%

Languages

110

- **French**: native speaker
- **English**: fluent (TOEFL iBT score of 105/120)
- **Japanese**: intermediate (level 2.5 of the JLPT)