Object Detection with Active Sample Harvesting

THÈSE N^{O} 7312 (2017)

PRÉSENTÉE LE 17 FÉVRIER 2017

À LA FACULTÉ DES SCIENCES ET TECHNIQUES DE L'INGÉNIEUR
LABORATOIRE DE L'IDIAP
PROGRAMME DOCTORAL EN GÉNIE ÉLECTRIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Olivier CANÉVET

acceptée sur proposition du jury:

Prof. P. Frossard, président du jury
Dr F. Fleuret, directeur de thèse
Prof. G. Blanchard, rapporteur
Prof. R. Sznitman, rapporteur
Dr M. Salzmann, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2017

Abstract

The work presented in this dissertation lies in the domains of image classification, object detection, and machine learning. Whether it is training image classifiers or object detectors, the learning phase consists in finding an optimal boundary between populations of samples. In practice, all the samples are not equally important: some examples are trivially classified and do not bring much to the training, while others close to the boundary or misclassified are the ones that truly matter. Similarly, images where the samples originate from are not all rich in informative samples. However, most training procedures select samples and images uniformly or weigh them equally.

The common thread of this dissertation is how to efficiently find the informative samples/images for training. Although we never consider all the possible samples "in the world", our purpose is to select the samples in a smarter manner, without looking at all the available ones.

The framework adopted in this work consists in organising the data (samples or images) in a tree to reflect the statistical regularities of the training samples, by putting "similar" samples in the same branch. Each leaf carries a sample and a weight related to the "importance" of the corresponding sample, and each internal node carries statistics about the weights below. The tree is used to select the next sample/image for training, by applying a sampling policy, and the "importance" weights are updated accordingly, to bias the sampling towards informative samples/images in future iterations.

Our experiments show that, in the various applications, properly focusing on informative images or informative samples improves the learning phase by either reaching better performances faster or by reducing the training loss faster.

Keywords: Computer Vision, Machine Learning, Image Classification, Object Detection, Bootstrapping, Monte Carlo Tree Search, Importance Sampling

Résumé

Les travaux présentés dans cette thèse concernent la classification d'images, la détection d'objets et l'apprentissage automatique. Que ce soit pour entraîner un classifieur d'images ou un détecteur d'objets, la phase d'apprentissage se résume à trouver une frontière de décision optimale entre les classes. En pratique, les exemples d'apprentissage n'ont pas tous la même importance. Certains sont aisément classifiés, tandis que d'autres, proches de la frontière ou mal classés, sont ceux qui ont de l'importance. De même, les images d'où proviennent les exemples d'apprentissage ne sont pas toutes aussi riches en exemples informatifs. Cependant, la plupart des méthodes d'apprentissage sélectionnent les exemples et les images de manière uniforme, et leur accordent la même importance.

La ligne directrice de nos travaux a été de mettre au point des méthodes pour trouver efficacement les exemples d'apprentissage les plus informatifs, sans jamais accéder à la totalité de l'ensemble d'apprentissage.

Les méthodes que nous proposons se basent sur une organisation des données (images ou exemples d'apprentissage) avec une structure d'arbre qui en reflète les régularités statistiques, en regroupant les exemples "similaires" dans les mêmes branches. Chaque feuille de l'arbre contient un exemple d'apprentissage et un poids correspondant à "l'importance" de l'exemple, et les nœuds intérieurs contiennent des statistiques sur les poids de leurs descendants. Cet arbre est utilisé pour choisir les images ou les exemples, en appliquant une méthode d'échantillonnage, et les poids "d'importance" de l'arbre sont ensuite mis à jour, pour biaiser l'échantillonnage vers les images/exemples informatifs lors des itérations futures.

Les expériences menées montrent qu'en choisissant soigneusement les images ou les exemples informatifs, on améliore l'apprentissage, soit en atteignant plus rapidement de meilleures performances, soit en réduisant la fonction de coût plus rapidement.

Mots clefs : Vision par ordinateur, apprentissage automatique, classification d'images, détection d'objets, bootstrap, recherche arborescente Monte Carlo, échantillonnage d'importance

Acknowledgements

I would like to express my gratitude to my advisor François Fleuret for his valuable advice all along these four years as well as for always encouraging me to do sound work.

I would like to thank Pascal Frossard, Gilles Blanchard, Raphael Sznitman, and Mathieu Salzmann for agreeing to be part of my jury and for reading my dissertation.

Idiap is an amazing place where to work, and I would like to thank the System Team, Louis-Marie, Frank, Norbert, Bastien, and Cédric for their support with the IT infrastructure; Nadine, Sylvie, and Vincent, for their help for the conference and everyday administrative tasks; and also Corinne, Chantal, Valérie, and Vanessa from EPFL for their help with the final thesis formalities.

I would like to thank my workmates Leonidas, Charles, Cijo, James, and Tatjana for fruitful discussions and valuable suggestions that helped me a lot.

Obrigado a ti, Christiane, por ter me apoiado durante esses quatro anos.

My work was supported by the Swiss National Science Foundation under grant 200021-140912 – DASH.

Martigny, August 2016

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

This chapter is an introduction to the main motivations of this work. The detailed description of the methods presented in this dissertation and accurate definitions are postponed until the next chapters.

1.1 Computer Vision

Computer vision aims at teaching machines how to see. The main workflow of any vision system is essentially to acquire an input signal, which is one or more images, to forward it to the machine designed to a specific task, and finally to get an information from the machine about the input signal.

Surprising as it may seem, it is difficult to design a vision system able to describe or analyse the content of an image or a video, whereas this can be considered an easy task for a human, including a child. We, as human beings, are very good at recognising objects in images even when the image is of poor quality or when the objects lie in a cluttered background. It may occur that the machine hallucinates objects in some places, while a human would have never made such a mistake.

Computer vision faces many challenges, in particular the one of designing methods that generalise well on unseen data and that are robust to small perturbations of the input signal. Another challenge is the efficient implementation of the methods. One can quickly come up with solutions to a problem that in practice happens to be intractable and unusable. And with the recent growth of data collections, another major challenge of computer vision is to handle large-scale databases to train the machines.

There are many real-world applications of computer vision that have driven researchers and engineers to develop sound mathematical approaches and design practical systems. And despite the fact that computer vision is a difficult task as previously stated, these systems have now reached a good level of performance to be deployed in industrial systems and everyday life objects.

One successful and widely used application is optical character recognition (OCR) which aims at recognising text from an image, be it typed or handwritten. OCR is used to automatically read ZIP codes and addresses on envelops to sort mail by postal destinations, to automatically read plate numbers at the entrance of car parks or to digitise the text of scanned documents to make them searchable.

Vision systems are also designed to assist people with disabilities, such as tetraplegia, brain injuries or sclerosis, to allow them to use computers and to communicate with others. Such systems can consist, for example, of an eye tracker which estimates where the user is looking at on the screen (*i.e.* the pointer of the mouse) and detects eye blinking (*i.e.* a mouse click) so that (s)he can navigate or enter text on a visual keyboard.

Recently, it has been announced that Cisco is currently measuring the street traffic of several squares in Paris. The project is part of a program of renovation of seven major squares and aims at better understanding the frequentation of the public space, namely counting the number of pedestrians, bike riders, cars and trucks, and building a map of the circulation. Several cameras and noise sensors are constantly recording the traffic and the data are sent to data centres for processing.

The work we are going to present in this dissertation lies in the subdomain of computer vision known as object recognition, which in particular consists of image classification and object detection. The former aims at assigning a label to an image (*i.e.* "Is this object in the scene or not?") while the latter focuses on localising all the instances of a given object in an image (*i.e.* "Is the object present? If so, how many times, where and at what scale?").

At some point, all techniques of image classification and object detection make use of a machine learning technique. The following section gives a brief overview of these methods.

1.2 Machine Learning

Machine learning is a field of artificial intelligence that aims at building predictors to infer the regularities of the data. In practice, the predictors are built with a training set, that is a set of examples which the learning procedure has access to. Training a predictor is done by minimising a loss function which reflects how well the predictor does on the training examples. The procedure keeps looking at the samples and adjusting its parameters until a convergence criterion is reached. Then, the predictor is evaluated on the test set, that is examples unseen during the training phase.

In practice, it is extremely important that the test samples be not seen during the training phase for fear of over-fitting and poor generalisation of the trained predictors. Sometimes, a validation set is used, which can be seen as an intermediate test set. The validation set can be built by splitting the original training set into two parts, one actually used for training, and the other one as the validation one.

In this work, we will also use a technique called Monte Carlo tree search, which is a type of reinforcement learning. In this setup, an agent explores an environment and gets feedback from it in the form of a reward. And its purpose is to maximise the long term reward by making the best moves or selecting the best actions, that is the ones that are more likely to yield a higher cumulative reward on the long term. Reinforcement learning therefore involves finding a trade-off between exploration of the environment and exploitation of the best identified areas of the environment.

1.3 Motivations

The first idea motivating our work is that in computer vision, we never train with “all the images in the world”. The largest computer vision image data sets contain several million images, out of the several trillion of the Internet. At some point, there always is a selection process, even if it is implicit. In practice, a human crawls the web, and eventually uses criteria to find “better” or more “useful” images. So training computer vision predictors is always based on a selection process, and we would like to do so in a “more” principled manner.

Moreover, the training process of machine learning methods in computer vision requires the use of training samples that in practice, are not all equally informative. Some samples could be qualified as “easy” because they are trivially classified and do not bring much to the learning phase, while other samples could be qualified as “hard” because they truly influence the learning phase and the decision process. These difficult samples are the ones needed for an efficient learning phase. Furthermore, the population of samples is usually structured. Some samples are similar and share common attributes, and the classifiers behave the same way on them. Therefore, the whole population of samples can be viewed as an ensemble of groups of “easy” samples, and groups of “hard” samples.

The main objective of our work is to leverage the consistency of the data during the training phase, to avoid looking at all the samples, and to accelerate or improve the learning.

While traditional methods tend to consider all samples equally and to visit all of them, be they “easy” or “hard”, our work investigates several strategies to get information on a few samples, to be able to discard others without looking at them, and thus to concentrate on the most important ones.

The following sections describe in more detail the context and motivations of the strategies presented in this dissertation.

1.3.1 Dense scanning of a sliding-window detector

As we will describe more thoroughly in section 2.2, a sliding-window detector densely scans an image, to predict at all possible locations whether the object of interest is present or not. An object detector is usually trained by constructing a sequence of predictors of increasing performance, each one trained with a fixed collection of positive samples (the object of interest) and a collection of so called hard negative samples that fool the previous predictors. This process is called bootstrapping.

A hard negative sample is a false positive sample, that is a negative sample incorrectly classified as a positive one, or in some cases, a negative sample that lies in the margin of the predictor. Hard samples are collected from a set of background images which do not contain the object of interest. So when the current predictor detects the object at a given location, the training set is augmented with this additional sample. Such an approach enriches the training set with negative samples that get closer and closer to the boundary between the positive and the negative populations, which are the ones that matter for a discriminative criterion.

One observes that within an image, the predictor's responses are correlated with other responses at nearby locations, suggesting that the decision at a given coordinate could be inferred from the neighbour decision(s). This can be accounted for by the smoothness of real-world images. Therefore, during the collection of false positives, some positions could be discarded without further computation given an observation in their vicinity. This problem is addressed in chapter 3.

1.3.2 Uniform selection of background scenes for bootstrapping

The other observation that we can make regarding bootstrapping is that in practice, hard samples tend to appear in a structured manner. Certain types of images contain patterns that generate false positives more or less frequently. Large uniform patches such as skies or empty walls can be ignored, while high frequency parts such as trees or bookshelves should be looked at carefully.

Moreover, image data sets usually have a structure by the way images were collected. Yet, all the current detectors are trained by uniformly selecting the next background image in which to search for hard samples.

To address this issue, we propose in chapter 4 to organise the image collection to reflect the statistical regularities of the images in order to better select them to train object detectors. In the end, many images are identified as non-informative and are not even looked at during training.

1.3.3 Careful selection of samples

With the growing need of data to improve classification and detection performance, data sets have become very large. The COCO data set (Lin et al., 2014) contains more than 100,000 annotated images, ImageNet (Russakovsky et al., 2015) several million, and these cannot fit in memory for learning applications.

The workaround to handle these data sets is to train predictors in an online fashion. While batch algorithms require all the data to be in memory for training, online algorithms process the training samples one at a time or by mini batches, which then becomes trackable. Most of the learning methods uniformly select the samples in the training set to update the predictor, while in practice only a fraction of them truly matter during learning. Indeed, it is because most of the samples are far from the decision boundary and thus are already well classified. And there are several clues indicating that all samples are not all equally important.

In the case of support vector machines trained in the dual space, this translates into a solution which is a linear combination of the kernel values between the sample to classify and a few training samples (the support vectors). The other training samples simply do not influence the final prediction, and would not be needed for training.

In the case of boosting, the difference of importance between samples translates into higher boosting weights for a few misclassified training samples. After a few iterations, many samples have a very small weight and no longer contribute to the selection of the next weak learners.

Despite this well known state of affairs, only few works have investigated how to prioritise the selection of the training samples. However, they do not use structures given *a priori* over the said samples, combining it with statistical observations over those already observed, to reject groups without looking at them. To address this problem, chapters 5 and 6 of this thesis propose to leverage the structure of the training set to efficiently concentrate on informative samples.

1.4 Contribution of this work

Following the room for improvement described in the previous section, we present our main contributions and how the dissertation is organised.

We first start, in chapter 2, by presenting the main tools that we are using in this work, such as the image classifiers, the object detectors, and the image data sets. We also describe in detail the training process of classifiers and detectors and how they are evaluated.

Chapter 3 deals with the observation that the detector's responses are correlated in the neighbourhood of a position, and presents an efficient strategy to collect false positive samples within images. We derive a simple affine model that is able to discard a neighbourhood around a visited location, without further computation, and without missing any false positive samples.

At the image collection level, chapter 4 presents a strategy to organise the images in such a way that it enables to make a decision on an entire group of images, either to choose them or to discard them, without further computation. The selection strategy is based on Monte Carlo tree search.

The following two chapters are concerned with image classification, and more precisely with the careful selection of samples and how to modulate their contribution during learning. Chapter 5 introduces an Importance Sampling Tree which allows to select more important samples, while compensating for the induced bias. And finally, chapter 6 introduces an Active Sampling Tree designed for support vector machines, to select the best samples for training, as well as dynamically update the tree for future selection.

We recall that each contribution follows the underlying theme of training a model without looking at all the samples: this is implemented by acquiring information on a subpopulation of samples, to precisely identify which samples are the most informative ones, and which ones can be discarded without being considered.

2 Background on Object Detection and Image Classification

Contents

This chapter presents some background on object detection and image classification. We review the various object detectors and image classifiers that we use in this thesis and describe the underlying ideas that make them work in practice. We also presents the various image data sets used in our experiments.

Whether it is image classification or object detection, the learning process consists, at some point, in training an image classifier. In the former case, the classifier is learnt to discriminate the classes against the others, while in the later case, it is trained to discriminate the object from the background. We therefore start by presenting image classification in more detail, then object detection, and finally some image data sets.

2.1 Image Classification

2.1.1 Objective

The objective of image classification is to automatically predict a discrete value which depends on the semantic content of an image. Two possible applications can be object classification, in which classes may be “horse”, “car” or “plane”, and scene categorisation, where classes may be “kitchen”, “outdoor” or “landscape”.

Figure 2.1 shows an overview of image classification in its most basic form. Usually, the input image I is mapped to an other space, called the feature space. The purpose is to get a common representation between all the images to be classified, but also to get a more abstract representation of the image that is suitable to learning algorithms. The image feature, which we call x, can then be passed to a classifier to make a prediction.

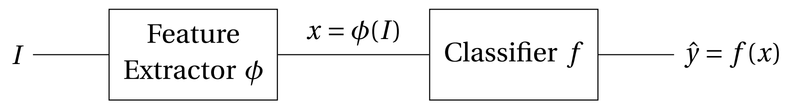


Figure 2.1 – Simplified overview of an image classification pipeline

Two major trends can be found in image classification. The first one is when the feature extraction is handcrafted by the human. Features are designed to a particular purpose in order to be discriminative (SIFT, SURF, HOG, etc.). A classifier is then trained with these features as input. The second trend of classification is when both the features and the classifier are learnt together, which in particular are (convolutional) neural networks. The latter tends to be more general because it learns everything in a sound framework, but has long remained very costly to train, until modern CPU and GPU reached TFLOPS (tera floating points operations per second).

2.1.2 Bag-of-words for Image Classification

This method has been adapted in computer vision for image classification (Csurka et al., 2004; Fei-Fei and Perona, 2005) or visual categorisation. Images are represented by a set of visual words (*i.e.* local features) coming from a dictionary (or codebook). The codebook can be any local feature, such as scale-invariant feature transform (Lowe, 1999) or local patches (sub-images). The feature vector describing the image is then passed to a common classifier. Yet again, classification is performed on this orderless collection of visual words. The importance of a code or its absence in the image will contribute differently depending on the class.

Although very good results are achieved, spatial information is inherently lost by the histograms. To overcome this drawback, several works proposed to take into account the spatial layout of the visual words (spatial pyramid matching, Grauman and Darrell 2005; Lazebnik et al. 2006), a process called pooling. Basically, the main idea is to aggregate the contributions of local features over predefined sub-regions of the image. An image is thus divided in several sub-regions, and the histograms of codes are computed on all these regions. This tends to add some localisation in the description of the image.

The pooling operation can refer to the mammalian primary visual cortex V1 (studied on cats by Hubel and Wiesel 1959). This stage in visual recognition of mammals is known to detect edges and pools them in local receptive fields, which are sensitive to the same kind of edges. Other stages follow and detect more complex patterns.

2.1.3 A single-layer network

This section presents the image classification pipeline proposed by Coates and Ng (2011) that we use in our work.

The pipeline of the feature extractor is depicted in figure 2.3. The input image is split in a set of 6×6 patches which constitute the local features. The patches are normalised by applying a whitening transformation that aims at decorrelating the pixels with one another. Then, given a codebook D of K elements (*i.e.* visual words, depicted on figure 2.2), each image patch is softly assigned to each visual words in the following way:

$$\forall\ 1 \leqslant k \leqslant K,\ f_k(p) = \max\left\{0, d_k^T p - \alpha\right\}, \tag{2.1}$$

where p is the patch to encode, d_k a visual word and α a fixed threshold. Each patch p activates the closest codes of the dictionary.

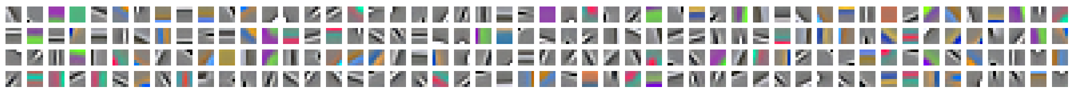


Figure 2.2 – Examples of visual words learnt in an unsupervised manner

The activations are then pooled on a regular 2 × 2 grid. All the contributions of a given code in a sub-region are aggregated in one new single score. This operation is known to create invariance over transformations such as translation. Another practical reason is that it reduces drastically the dimension of the feature vector. Basically, pooling is performed by averaging the activations of codes in the neighbourhood.

The codebook is learnt in an unsupervised manner by uniformly extracting patches from the training images and running a clustering algorithm. Here, spherical k-means is used and

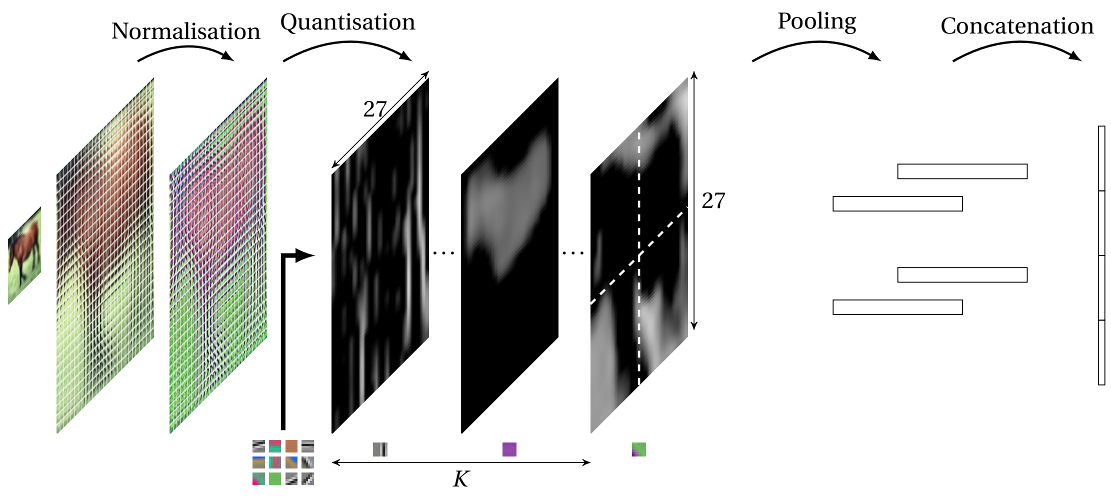


Figure 2.3 – Single-layer pipeline of Coates and Ng (2011)

computes a dictionary D by solving the following optimisation problem:

$$\begin{aligned} \min_{D,s} \quad & \sum_{i=1}^{N} ||D^T s_i - p_i||_2^2 \\ \text{subject to} \quad & \forall\, 1 \leqslant i \leqslant N,\ ||s_i||_0 \leqslant 1 \\ \text{and} \quad & \forall\, 1 \leqslant k \leqslant K,\ ||d_k||_2 = 1, \end{aligned} \tag{2.2}$$

where s_i is a code associated with the input patch p_i and $D = [d_1, \ldots, d_K]^T$ is the learnt dictionary of codes d_k. The first constraint means that each input patch p_i is associated to at most one non-zero entry. The second constraint means that all learnt codes are normalised to unit length. Finally, a linear classifier is learnt on top of this feature extractor.

Figure 2.4a shows the training and test accuracies obtained on the CIFAR10 data set (described in section 2.3) as a function of the codebook size. Larger dictionaries result in better performance as the classifier is able to better discriminate the samples.

Another interesting parameter is the number of iterations during the unsupervised phase (k-means). The spherical k-means algorithm is ran with simple white noise patches as initial solution. As the algorithm runs, it learns better codes: over the iterations, the dictionary drifts from noised patches (iteration 0) to meaningful sharp edges as depicted in figure 2.2.

Figure 2.4b shows the test accuracy as the number of iterations of k-means increases (run to learn $K = 800$ codes). Surprisingly, although the codebook consists of noisy patches at iteration 0, the resulting accuracy is already 70%, and after only 3 iterations reaches 76%. This result is very surprising because it means that the pipeline is able to learn with a meaningless dictionary. This experiment supports the results presented by Saxe et al. (2011) and Jarrett et al. (2009) where the authors noticed that some pipeline architectures could perform very well with no training (choosing random weights for instance). One explanation given is that

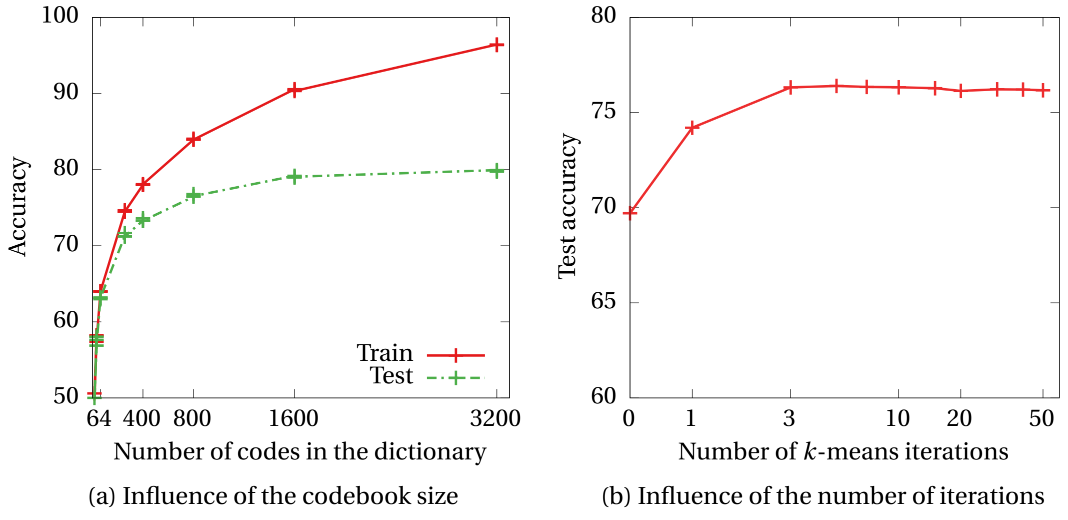


Figure 2.4 – Analysis of the single-layer classifier

some architectures are inherently frequency selective and translation invariant.

To summarise, the basic principle of image classification is to map an input image to a feature space and make a prediction on the class of the image. This component is the base concept in object detection that we now present.

2.2 Object Detection

Object detection aims at localising the position and the scale of all the instances of a given object of interest in an input image. Figure 2.5 shows an example of the output of a face detector. The goal is to locate all the faces present in the image (green detections) while avoiding false alarms (red boxes). Sometimes, the detector misses some targets (blue boxes), usually because of occlusion (the object is not entirely visible), part occlusion (glasses, hat, etc.) but also because the object appears in an unusual pose (rotation).

2.2.1 Sliding-window object detectors

Paradigm

A sliding-window object detector consists of a binary classifier that is trained to discriminate the object of interest from the background. The classifier takes as input a fixed size image (*e.g.* 80×80) and outputs a score related to the presence or absence of the object of interest.

The classifier is then densely applied at all positions and all scales of the image to find all the instances of the object. When multiple detections occur in the same vicinity, they are removed by non-maxima suppression to keep only one of them.

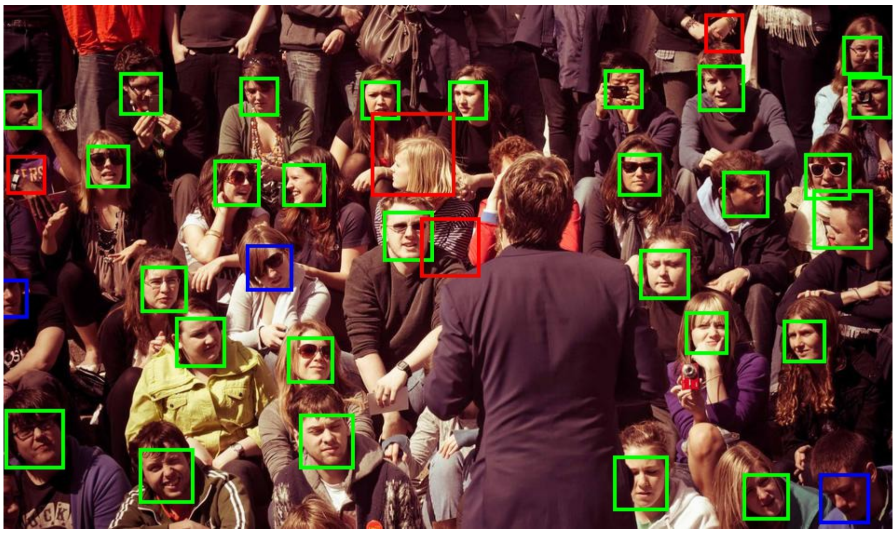


Figure 2.5 – Result of a face detector. Green boxes are correct detections, blue are miss detections, and red are false alarms.

Training

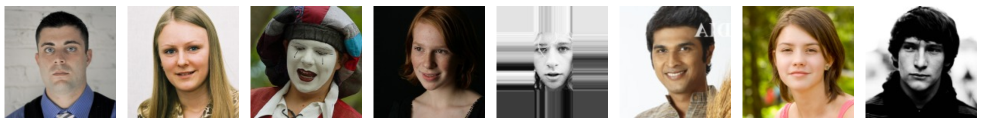
The core component is thus the binary classifier. It is trained through a bootstrapping procedure that aims at building a series of classifiers of increasing performance. Initially, the classifier is trained with a population of positive samples (see figure 2.6a in the particular case of face detection) and negative samples (see 2.6b). At this point, it already discriminates well between objects and background, but still raises too many false alarms to be used.

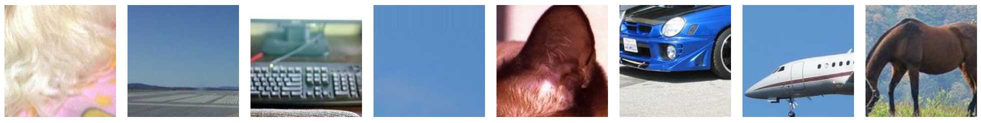
The bootstrapping approach (or "hard negative mining") consists in enriching the training set with these false positive samples, also called hard samples. In practice, the current classifier is applied on object-free images: if the classifier has a positive response at a given location, the corresponding sample is a false positive since no object is in the image, and it is added to the training set. The classifier is eventually retrained with this augmented training set. Figures 2.6c and 2.6d show examples of hard samples after the first and second bootstrapping step respectively. As the training goes on, false positives become harder and harder and force the classifier to differentiate faces from face-like shapes.

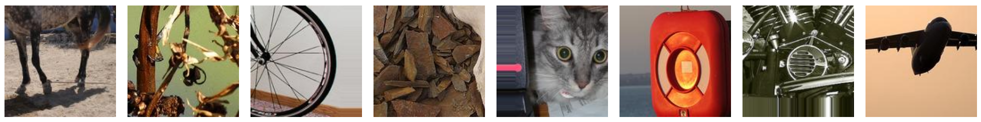
2.2.2 Viola and Jones detector

The Viola and Jones face detector (Viola and Jones, 2004) is a sliding-window detector whose binary classifier is a strong learner $H(x)$ made of the linear combination of several weak learners:

$$H(x) = \sum_i \alpha_i h_i(x), \tag{2.3}$$


(a) Positive images


(b) Negative images


(c) Hard images after bootstrapping step 1

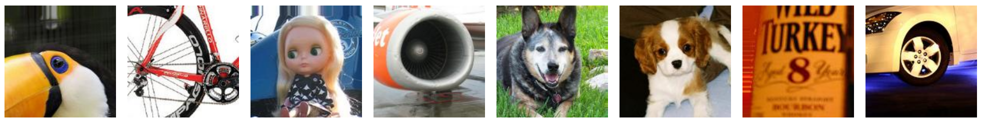

(d) Hard images after bootstrapping step 2

Figure 2.6 – Training images for face detection

where x is the input image window to classify as object or background, h_i is a weak learner, and $\alpha_i \in \mathbb{R}$ its weight. A weak learner is a classifier that has a poor classification accuracy, but the combination of several of them yields an accurate classifier.

The weak learners are combined through a boosting algorithm, which aims at building a strong classifier of increasing performance, by carefully selecting the next weak learner h_i and its weight α_i, so as to reduce the classification error on the training set.

In the Viola and Jones detector, the weak learners are simple stumps consisting of two parts: the first part is a Haar wavelet (see figure 2.7) which computes the difference between the sum of the pixels in the black areas with the sum of the pixels in the white areas, and the second one is a threshold function which outputs +1 if the difference is above a constant, and -1 otherwise.

Moreover, to increase the speed of the detector, the strong learner is trained in a cascade fashion, such that when some partial sums fall below a threshold, the window is rejected. In this way, some computation is saved.

2.2.3 Aggregated Channel Feature detector

The aggregated channel feature detector (ACF) was introduced and improved by Dollár et al. (2009, 2010, 2014) and is based on the Viola and Jones detector that we have previously described. The ACF detector consists of richer features (see figure 2.8). An RGB image is

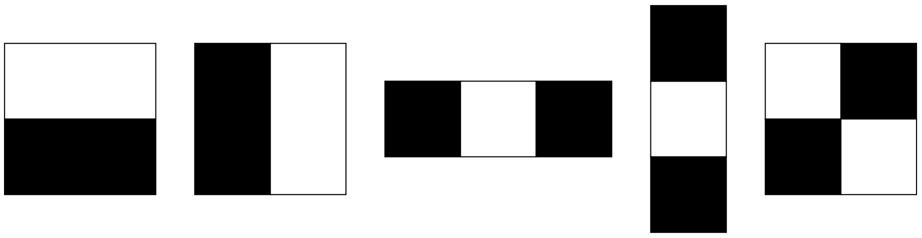


Figure 2.7 – Haar wavelet patterns to compute features

decomposed in several grey level channels representing colour (in LUV space, 3 channels), gradient magnitude (1 channel) and several histograms of oriented gradients (6 channels, for 6 orientations). The channels are pooled by 4 × 4 blocks, so a 32 × 64 image results in a ten 8 × 16 channels, for a total of 1280 features.

In this framework, the weak learners are shallow trees (depth from 2 to 5) over the features. As a comparison, the stumps of the Viola and Jones detector are one-depth trees. Moreover, Dollár et al. (2014) demonstrate that the ACF features can efficiently be interpolated at nearby scales, which further reduces the time of computing the features.

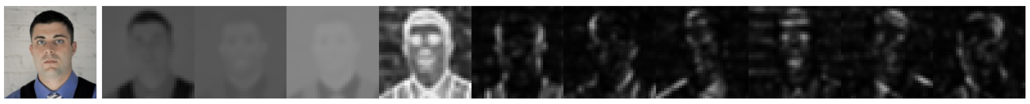


Figure 2.8 – Illustration of the aggregated channel features

2.2.4 Deformable Part-Based Model

Dalal and Triggs introduced the histograms of oriented gradient (HOG) in the context of pedestrian detection (Dalal and Triggs, 2005). The HOG descriptors are computed by aggregating the orientations of the gradient weighted by its magnitude on a regular grid of typically 8 × 8 pixels. The detector uses support vector machines to classify the windows in either object or background.

The Dalal and Triggs detector has later been extended by Felzenszwalb et al. (2010b) with the deformable part-based model (DPM). This detector enriches the previous one by using additional parts at a finer resolution, and by still using HOG as a descriptor. The model consists of a root template similar to the original detector and of several parts that are able to move from the the root, accounting for the deformation of the object. Figure 2.9 shows an example of a DPM face detector: the first row contains the roots of the components, and the second the parts.

As the DPM is computationally expensive because of all the convolutions that are made at several octaves of the pyramid, several works have investigated various strategies to accelerate the evaluation of the model on an image. Dubout and Fleuret (2012) have proposed to perform the convolutions in the Fourier domain that leads to an order of magnitude speedup. Dean

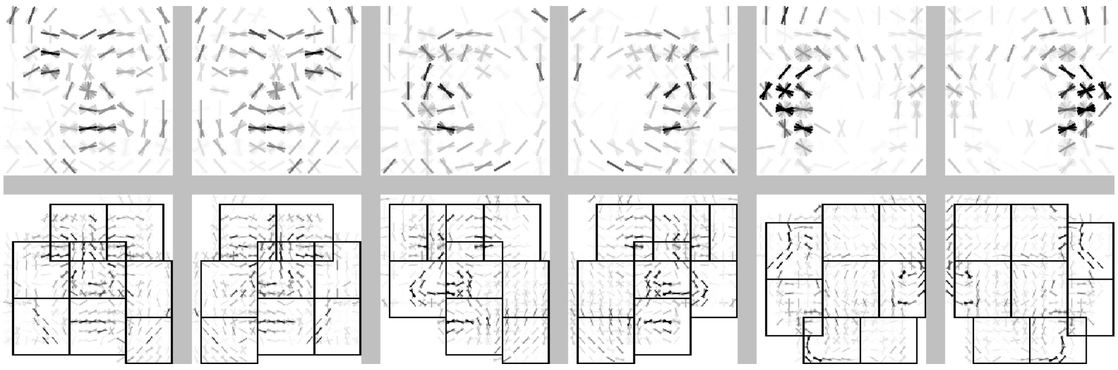


Figure 2.9 – Illustration of a face deformable part-based model

et al. (2013) have used hashing techniques to replace the convolutions, and have shared the parts between various objects, which allows to test 100,000 detectors on a single machine. And Yan et al. (2014) have combined various strategies to obtain a real time DPM model: they proposed to use a cascade and a low rank root filter to perform the convolutions faster.

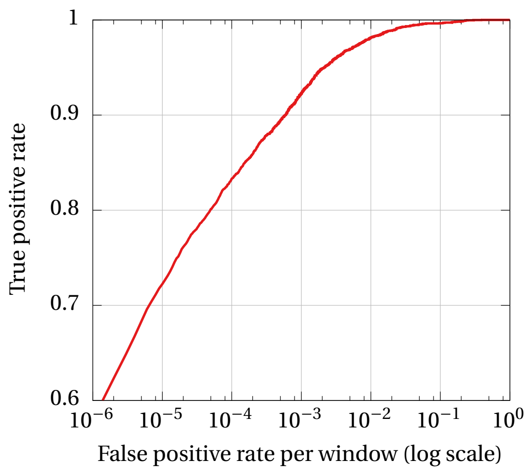
2.2.5 Evaluation of object detectors

The quantitative evaluation of any system is important especially to compare systems with one another. To this purpose, a metric should be designed to reflect the performance of the system.

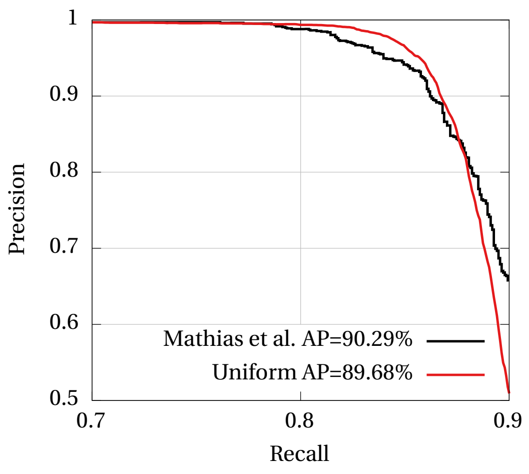
In the case of object detection, evaluation boils down to measuring the performance of the binary classifier trained to discriminate the object from the background. When the classifier does not make any mistakes, it either correctly classifies a positive sample as positive (true positive) or correctly classifies a negative sample as negative (true negative). When the classifier makes a mistake, it either incorrectly classifies a negative sample as positive (false positive, type I error) or incorrectly classifies a positive sample as negative (false negative, type II error).

It is difficult to minimise both error types at the same time when designing a system. Therefore, a trade-off is made depending on the application. In some cases, it is imperative not to have any false positives, such as when classifying enemy targets (positive class) versus non-enemy (*e.g.* hospital, school, etc.). In other cases, false negatives should be avoided at all costs, such as when classifying disease patient (positive class) versus healthy people. In the latter case, beside the psychological side effect of being incorrectly classified as sick, further tests can later be performed to nullify the first diagnostic, but it is of the essence that a disease patient never be classifier as healthy.

To evaluate the performance of the classifier, we can use a receiver operating characteristic curve (ROC) which represents the true positive rate as a function of the false positive rate (see figure 2.10a). To make such a curve, we extract positive windows which contain the object of interest, and negative windows, and the detector is applied on them.



(a) ROC curve, ideal is top-left



(b) Precision-recall curve, ideal is top-right

Figure 2.10 – Evalution of a binary classifier

An alternative to ROC curves is precision-recall curves (see figure 2.10b). In this case, the detector is applied on the test images and we compute the precision P and the recall R with:

$$P = \frac{TP}{TP+FP} \quad \text{and} \quad R = \frac{TP}{TP+FN}, \tag{2.4}$$

where TP stands for true positive, FP for false positive and FN for false negative.

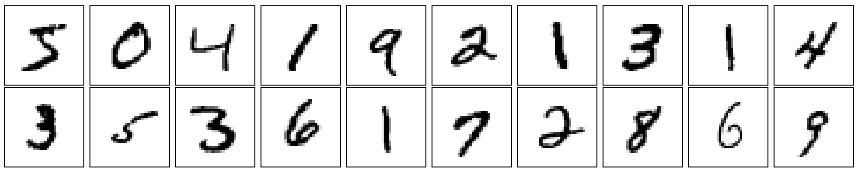
As these measures involve two parameters, it is difficult to obtain an ordering of different methods. To this purpose, we usually use the area under the curve (numbers in the key of figure 2.10b). However, this score can be confusing because as we can see on figure 2.10b, although the red curve has a smaller area (89.68%) than the black one (90.29%), the red curve is above the black one in the range $0.8 < R < 0.85$, which would be the use case of the detector for real applications.

2.3 Image data sets

2.3.1 Image Classification data sets

The MNIST data set (LeCun et al., 1998) is a collection of 28×28 grey scale handwritten digits, which has been extensively used in the field of computer vision and machine learning to train image classifiers. It consists of 60,000 training images and 10,000 test images belonging to 10 classes (digits from 0 to 9). Figure 2.11a shows a few examples of the MNIST images.

This data set has been augmented with artificial distortions yielding the InfiMNIST data set (Loosli et al., 2007). The distortions consist of sub-pixel elastic transformations (Simard et al., 1992, 1998) alongside with additional translations of one pixel. A distortion is the result of


(a) Images from the MNIST data set

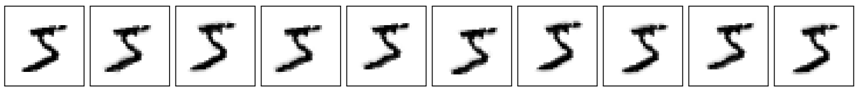

(b) First training sample from InfiMNIST and its first 9 distortions

Figure 2.11 – MNIST and InfiMNIST images

adding to the original image the componentwise product between a tangent vector in one direction and a deformation field such that

$$I_{\text{distorted}}(i,j) = I(i,j) + \alpha\left[f_x(i,j)\, t_x(i,j) + f_y(i,j)\, t_y(i,j)\right]. \tag{2.5}$$

t_x (resp. t_y) is the tangent vector of image I in the horizontal (resp. vertical) direction, i and j are the coordinates of the pixels, and f_x (resp. f_y) is the deformation field used in the x (resp. y) direction.

In practice, the distortions can efficiently be computed on the fly by storing the original samples, the precomputed tangent vectors of each original and some precomputed smoothed distortion fields. By combining all the distortion fields in both direction with translations, the InfiMNIST data set can provide several trillion images. Figure 2.11b depicts the first training image of MNIST alongside with its first nine distortions.

The CIFAR10 data set (Krizhevsky and Hinton, 2009) is a collection of 32×32 RGB images taken from the "80 million tiny images" collection (Torralba et al., 2008). The training set consists of 50,000 images and the test set of 10,000 images belonging to 10 classes of objects such as "airplane", "cat", "dog", "car" or "horse". Figure 2.12 shows some images of this data set.

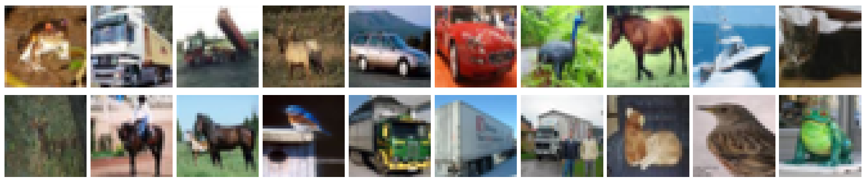
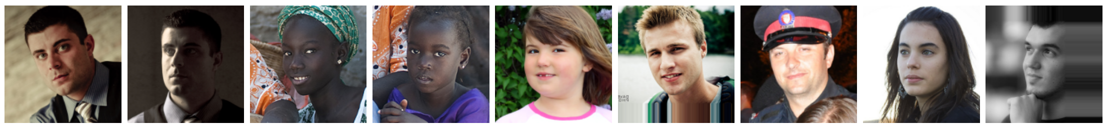


Figure 2.12 – Images from the CIFAR10 data set

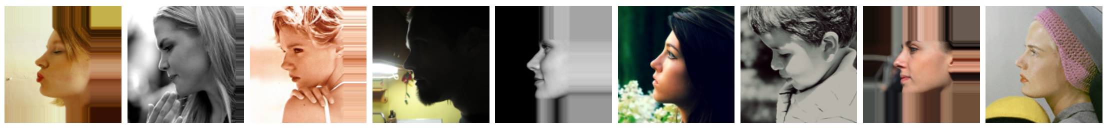
2.3.2 Face data set

The annotated facial landmarks in the wild (AFLW, Koestinger et al. 2011) is a large collection of annotated faces in real-world images. It consists of more than 21,000 images for a total of

(a) Images used to train a frontal component (0 ⩽ |yaw| ⩽ 20)



(b) Images used to train a semi profile component (20 ⩽ |yaw| ⩽ 60)



(c) Images used to train a profile component (60 ⩽ |yaw| ⩽ 100)

Figure 2.13 – Cropped images from the AFLW data set to train a face detector

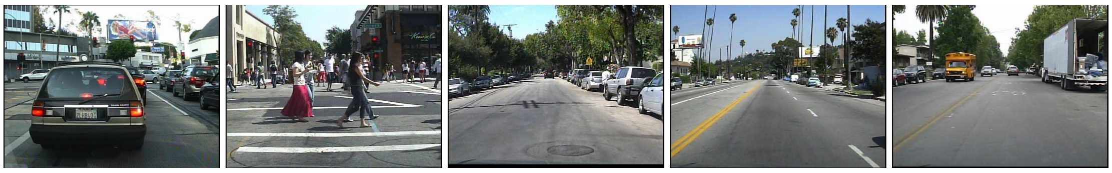
around 25,000 faces. Each face was manually annotated with several facial landmarks. From the annotations, the Euler angles of each face can be determined: the roll, the pitch and the yaw, that is the orientation of the face in 3D.

Face detectors are trained by cropping the objects of interest and training a classifier to discriminate the object from its background. The object is usually cropped with some additional context to avoid over-fitting. Figure 2.13 shows some examples of faces from the AFLW data set. As was done by Mathias et al. (2014) and Yang et al. (2014), the faces are split in three groups based on their yaw to train three separate detectors: a frontal, a semi-profile and a profile component. To regularise a bit the data set, faces with a pitch and a roll larger than 22.5 are ignored. In the end, there are 6,754 crops for the frontal component, 6,784 for the semi-profile and 3,182 for the profile component. The artefacts observed in several images correspond to the padding added when the ground truth goes outside the image.

2.3.3 Pedestrian data sets

The INRIA Person data set (Dalal and Triggs, 2005) is a collection of photos gathered to train pedestrian detectors. It consists of 2,416 128×64 images of pedestrians and 1,218 pedestrians-free scenes for training. The testing set consists of 1,132 pedestrians and 453 scenes.

The Caltech pedestrian data set (Dollar et al., 2012) consists of 10 hours of 30 Hz videos (250,000 frames of 640×480 pixels) shot from a moving vehicle in a city. This data set is challenging because the quality of the frames is not so high compared to static pictures and it contains a large amount of frames. The data set is split into short sequences of a few minutes, each being shot in various areas, such as airports, cities or suburbs. The training set which contains 128,000 images is usually sub-sampled down to 4,000 images (Dollár et al., 2014) or to 32,000


(a) 640×480 scenes from the training set

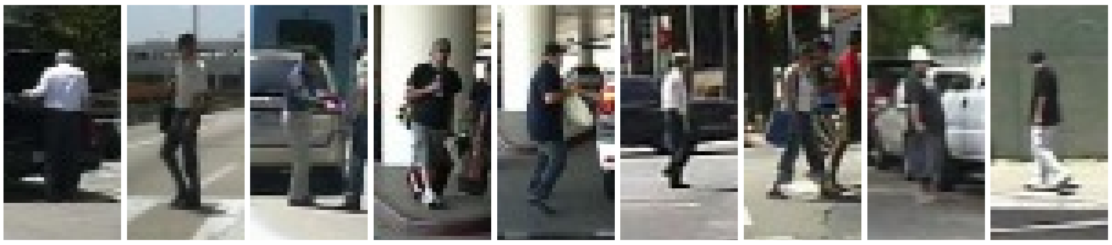

(b) 64 × 32 cropped positive samples from the training set

Figure 2.14 – Images from the Caltech pedestrian data set

images (Nam et al., 2014) and the usual test set contains around 4,000 images. Figure 2.14 shows some scenes taken from the sequences, and some cropped pedestrians used as positive samples to train pedestrian detectors.

3 Efficient Sample Mining for Object Detection

Contents

This chapter presents a way to drastically reduce the number of locations at which a detector should be applied to find hard negative samples. The main idea of the method is to discard the neighbour positions around a location where the detector has not found any hard samples. It is based on the following publication:

Olivier Canévet and François Fleuret. Efficient sample mining for object detection. In *Proceedings of the Asian Conference on Machine Learning (ACML)*, pages 48–63, 2014

3.1 Introduction

3.1.1 Sliding-window object detectors

Sliding-window object detectors, as presented in section 2.2.1, are trained through a multi-step procedure called bootstrapping, which consists in using the current step's detector to enrich the training set with hard negative samples. Bootstrapping is crucial to achieve good performance and to be able to use the detector in real-world applications.

In this chapter, f_1 denotes the initial classifier trained to discriminate between object and background, and f_2 the one retrained with hard samples (one step of bootstrapping). f_1 is simply trained with a population of positive samples (*i.e.* the object) and a population of negative samples (*i.e.* uniformly taken in the background of the object).

A hard negative sample x can thus be defined as a negative sample on which the binary classifier f_1 has a positive response, or in the case of SVM, that lies inside the margin of the predictor ($f_1(x) \geqslant -1$, as in Felzenszwalb et al. 2010b, see figure 3.1). Object-free scenes are thus densely parsed until enough samples have been found and the second classifier f_2 is trained on this augmented set.

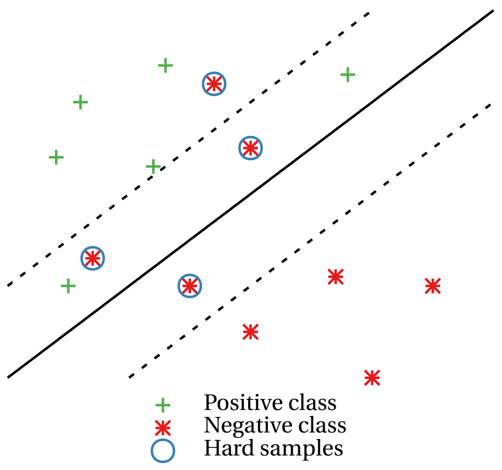


Figure 3.1 – Illustration of where hard samples lie compared to the frontier and margin

3.1.2 Related work

Many works over the past years have focused on proposing new features to improve the performance of the final detector. Haar wavelets proposed by Viola and Jones (2004) have shown to be extremely efficient combined with AdaBoost to detect faces in images. Histograms of oriented gradients by Dalal and Triggs (2005) have long been used by almost all detectors before being superseded by convolutional neural networks. Walk et al. (2010) proposed features based on the optical flow in videos and on the self-similarity between colour channels.

Finally, integral or aggregated channel features (Dollár et al., 2009, 2014) are both very fast and very efficient for pedestrian detection when combined with two-depth trees.

Other works have concentrated on accelerating detection at test time. Making use of cascades (Viola and Jones, 2004; Bourdev and Brandt, 2005; Felzenszwalb et al., 2010a; Fleuret and Geman, 2001, 2002) speeds up detection dramatically by discarding many windows in the early processing steps. Pedersoli et al. (2011) proposed a coarse-to-fine approach that reduces the number of locations to visit by discarding areas at smaller scales that will not be tested at larger scales. Gualdi et al. (2010) introduced a probabilistic framework that exploits the responses of the cascade to efficiently span the image at test time. Regarding deformable part based models, Dubout and Fleuret (2012) have proposed to compute efficiently the convolutions in the Fourier domain. The fastest detectors are now able to process around 30 frames per second which makes them practical for real-time applications.

The work by Henriques et al. (2013) shows that in the particular case of linear classifiers, it is possible to train with the virtual, fully translated versions of the negative samples, because the Gram matrix of the translated samples is block circulant and can be factorised in an efficient way, removing redundancies between samples. They thus show that they can avoid several steps of bootstrapping and yet achieve similar final performances.

3.1.3 Motivation

Bootstrapping is used in all the cited works (except the last one) independently of the nature of the classifier (SVM, boosting), of the detection framework (cascade or not), or of the features. It allows the process to concentrate computation on the most difficult samples and yields a usable sliding-window detector. Dalal and Triggs (2005) use one step of bootstrapping but add all the false positives that are found. Dollár et al. (2009) use two steps, while other works use more (Walk et al., 2010; Dollár et al., 2014) and can go up to 10 (Felzenszwalb et al., 2010b) but limit themselves to a few thousands samples added each time. When training cascades, bootstrapping is used at each new level of the cascade (Viola and Jones, 2004).

Despite the popularity of this approach, little work has been done on how to efficiently parse scenes to find hard samples during the bootstrapping step. The traditional approach sees bootstrapping as a particular case of detection in which one uses the non-mature classifier f_1 to increase the training set with difficult samples for f_1. In this chapter, we propose to leverage local consistency of f_1 in the image plan to limit computation to a sparse subset of locations.

Figure 3.2 shows the responses of a pedestrian detector at every other location in the image. The responses suggest a smoothness across the image and that, given a response at some position, we can infer the responses of the classifier in its neighbourhood.

In this context, we propose a simple yet efficient strategy that estimates the largest neighbourhood that can be discarded around a position and that does not contain a hard sample (section 3.2). This is faster than the traditional approach because it discards lots of positions,

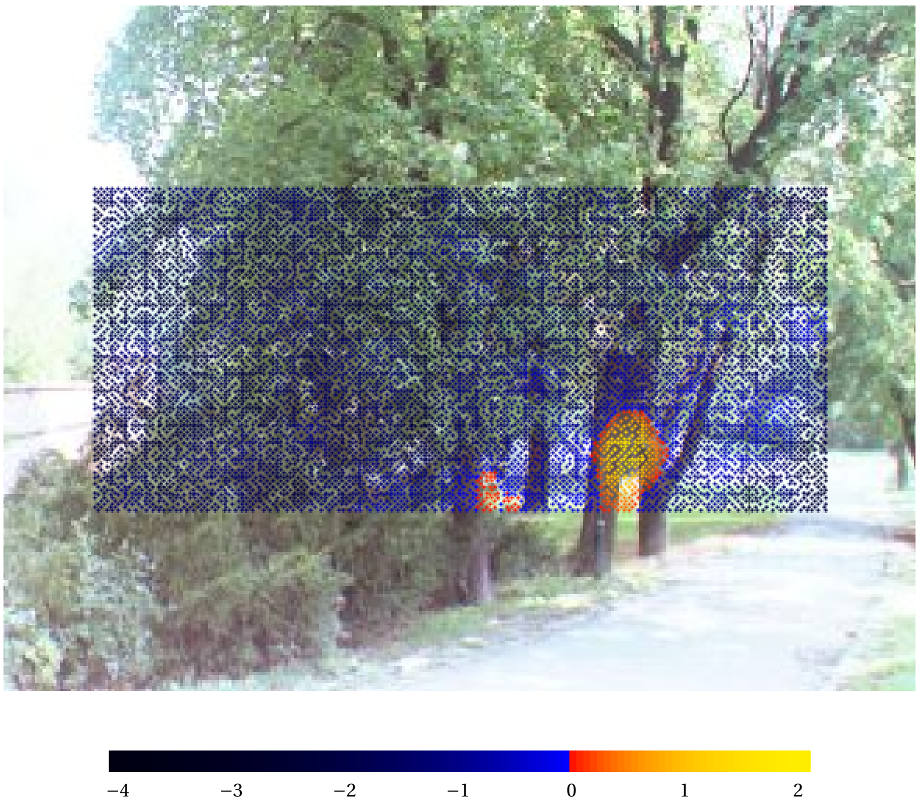


Figure 3.2 – Initial pedestrian detector responses (HOG features with linear SVM). The detector was applied every two pixels. One point on the image corresponds to the centre of the 128×64 sliding-window. The colour of the points corresponds to the value of the detector response. Here, the false positives are around the trees, the shape of which is reminiscent of a pedestrian.

but it is also as effective because it finds as many hard samples. We also investigate several methods to sample sequentially the scenes to process (section 3.2.2). We evaluate the method in section 3.3 on pedestrian and face detection, and show that it is able to find as many hard samples as the baseline while being significantly faster.

3.2 Proposed strategy

Our purpose is to efficiently build a subset $\mathcal{H}$ of the full set of hard samples $\mathcal{H}^*$. At this point, we assume that we have an initial binary classifier f_1, trained as described before, and a series of object-free scenes $S_n,\ n = 1, \ldots, N$. A sample x is considered hard if $f_1(x) \geqslant m$, where $m \in \mathbb{R}$ is fixed.

Given that we have already selected a series of hard samples, to select a new one, we first choose a scene S_n and then a location $u \in S_n \setminus V_n$, where V_n is the subset of discarded locations in S_n. We extract the sample x at location u and compute $f_1(x)$.

Scene	Image at a given scale in which negative samples are extracted
$\mathscr{L}$	Set of all the possible sample locations in the scene
f_1	The initial classifier
f_2	The classifier trained with the augmented set
x	A $h \times w$ sample on which we can apply f_1 or f_2
m	A threshold above which a sample is considered hard
hard sample	A sample such that $f_1(x) \geqslant m$
S_n	An object-free scene used to find negative and hard samples
V_n	The set of already visited or discarded positions in S_n
Exhausted	A scene is exhausted when all the locations have been visited or discarded

Table 3.1 – Notations and designations

If $f_1(x) < m$ (*i.e.* x is a true negative), V_n is updated with u as well as all locations within a circle of radius r^- centred at u, without actually computing f_1 on them. We call this *discarding the neighbourhood* (*i.e.* these positions will not be selected in the future). As explained in the following sections, depending on the method we consider, the radius r^- may be a constant, or a function of the response $f_1(x)$. Figure 3.3 shows examples of discarded regions for various radii. When all positions in S_n have been either visited or discarded, we say that the scene is *exhausted*.

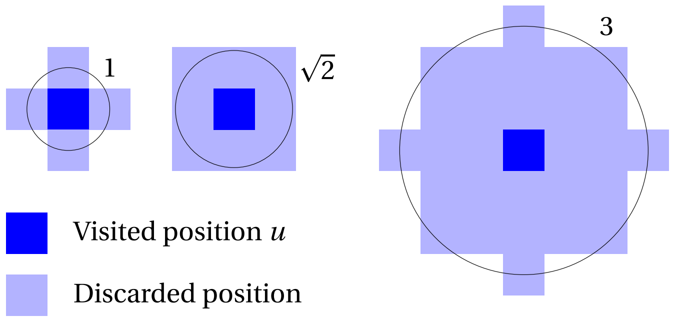


Figure 3.3 – Various discarding radii

If $f_1(x) \geqslant m$ (*i.e.* x is a false positive), we add x to $\mathcal{H}$, u to V_n, and we discard a small neighbourhood of radius r^+ around the false positive: we thus discard positions of redundant samples. In all the methods we present, r^+ is constant.

Finally we repeat the procedure by choosing a scene and a position until we have found enough hard samples or until all the scenes are exhausted.

It is important to note that we do not exhaust a scene before moving to the next scene. Rather, we pick one scene and a location at a time, so that we gradually visit them all. With small data sets, this is unimportant, especially if the purpose is to exhaust all the scenes (*i.e.* visit all locations). With very large data sets however, it is imperative to enforce the sampling to be distributed over all the scenes, to avoid generating a set of samples of limited diversity.

We now describe the strategy we propose to parse the scenes efficiently: the key idea is to learn the discarding radius r^- as a function of $f_1(x)$. We illustrate the rest of the section with pictures from the INRIA Person data set.

3.2.1 Strategy to discard non-visited locations in a scene

The novel methods we propose in the next sections rely on the relation between the point-wise response of the classifier at a certain location in the scene, and the presence of hard samples in its neighbourhood.

Without loss of generality, we assume that all the scenes have the same size and we define the lattice of extractible positions $\mathscr{L} = [\![1, H]\!] \times [\![1, W]\!]$[1]. A scene S is a random variable on $[\![0, 255]\!]^{\mathscr{L}^3}$.

For simplicity, given a position $u \in \mathscr{L}$ and a classifier f, we will use the notation $f(u)$ for the response of f on the sample extracted at location u in the scene.

Let U be a position sampled uniformly on $\mathscr{L}$, and R the distance between U and the closest false positive, that is

$$R = \min_{\substack{v \in \mathscr{L} \\ f_1(v) \geqslant m}} \|v - U\|_2 \tag{3.1}$$

Using $f_1(U)$, our purpose is to predict the *largest* neighbourhood that we can discard *without* discarding positions that actually contain hard samples, that is finding a lower bound on the random variable

$$R \mid f_1(U). \tag{3.2}$$

Figure 3.4 shows the empirical distribution of $(f_1(U), R)$: this plot was done on the INRIA Person data set with a HOG pedestrian detector. One dot represents the distance between a position u in the image and the position v of the closest false positive from u, as a function of the response $f_1(u)$ at position u. We see that for a given negative response $f_1(x)$, there is a non-zero lower bound: the closest false positive from u tends to be farther from a true negative x as $f_1(x)$ increases. By learning this lower bound, we can therefore discard a circular area around u with an *adaptive* radius $r^- = r^-(f_1(x))$.

The standard way to estimate the lower bound of the distribution is to extract the points near the bound and fit a model on them. Many models can be used as well as many ways to learn it: linear or quadratic model, step function, etc.

As we observe on the two data sets we used, the lower bound is properly modeled as an affine function of the classifier's response. This is the model we use for the *adaptive discarding*

[1]The size of the lattice is actually not the size of the scene: if we extract samples of size $h \times w$ in a $H_S \times W_S$ scene, the extractible region is $(H_S - h + 1) \times (W_S - w + 1)$

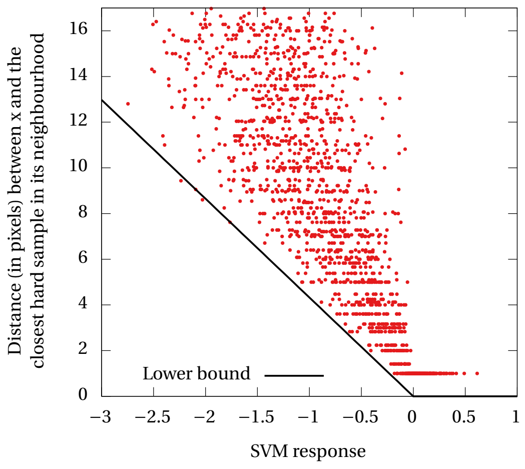


Figure 3.4 – Empirical distribution of $(f_1(U), R)$ as defined in 3.2.1 with $m = 0$. One green dot $(f_1(u), r)$ represents the distance $r = ||v - u||_2$ of the closest false positive (at location v) from sample x (at location u) as a function of the response $f_1(x)$ of the initial classifier. For instance, given a true negative sample x^- such that $f_1(x^-) = -2$, we are assured that the closest false positive is farther than ~8 pixels away, which allows us to discard a disc of radius 8 around u, that is ~200 locations that will never be chosen. The horizontal line for $f_1(x) \geqslant 0$ indicates that empirically, any false positive has a false positive neighbour.

radius:

$$r^-(f_1(x)) = -a \times (f_1(x) - m), \quad a \geqslant 0, \tag{3.3}$$

where m is the threshold above which x is considered hard. Before building $\mathcal{H}$, we draw the empirical distribution by selecting a few 30×30 sub-lattices out of hundred scenes and compute the distance to the closest hard sample at every location. We then estimate the optimal $a^\star$ by averaging over the 10 lowest values of

$$\frac{r(f_1(x))}{f_1(x) - m}. \tag{3.4}$$

3.2.2 Strategy to select a scene

In the works of Dalal and Triggs (2005) and Viola and Jones (2004), scenes are visited at all locations either until it no longer fits in memory or to keep a reasonable training time.

However, with the recent year large increase of the data sets, it becomes impractical to exhaust all scenes. We here present different methods to choose the next scene to visit.

Uniform sampling consists in choosing a scene S_n uniformly with replacement in the subset of non-exhausted scenes, that is with the distribution:

$$\forall\, 1 \leqslant n \leqslant N, \quad p(S_n) = \frac{1}{N} \tag{3.5}$$

Size sampling consists in choosing a non-exhausted scene according to its size, that is with the distribution

$$\forall\, 1 \leqslant n \leqslant N, \quad p(S_n) = \frac{1}{Z}|S_n|, \tag{3.6}$$

where Z is a normalisation constant and $|S_n|$ the number of extractible positions in the lattice of S_n.

Unexplored sampling consists in choosing a non-exhausted scene according to its unexplored area, that is with the distribution

$$\forall\, 1 \leqslant n \leqslant N, \quad p(S_n) = \frac{1}{Z}(|S_n| - |V_n|), \tag{3.7}$$

where $|V_n|$ is the number of discarded *or* visited positions in scene S_n.

Erf sampling consists in choosing a non-exhausted scene according to the classifier responses on already visited locations in the scene. Let μ_n and σ_n^2 be the mean and variance of the classifier responses in scene S_n at some point in the process. We sample according to the distribution

$$\forall\, 1 \leqslant n \leqslant N, \quad p(S_n) = \frac{1}{Z}\min\left(E_n, \frac{1}{\pi r^{+2}}\right)(|S_n| - |V_n|), \tag{3.8}$$

where

$$\forall\, 1 \leqslant n \leqslant N, \quad E_n = \int_m^{+\infty} \mathcal{N}(\mu_n, \sigma_n) = \operatorname{erfc}\left(\frac{x - \mu_n}{\sqrt{2}\sigma_n}\right), \text{ see figure 3.5} \tag{3.9}$$

This sampling is designed to select a promising scene, that is one which has already provided false positives will have a larger score. The min provides an upper-bound on the score to avoid having too large values of E_n. The constant $1/\pi r^{+2}$ corresponds to the number of samples we would collect if *all* the remaining positions were actually false positives.

3.2.3 Implementation detail for sampling

We may need to sample several million times, which can be computationally prohibitive if done in a naive way. We here provide details of the implementation to select the scene in which to extract the sample.

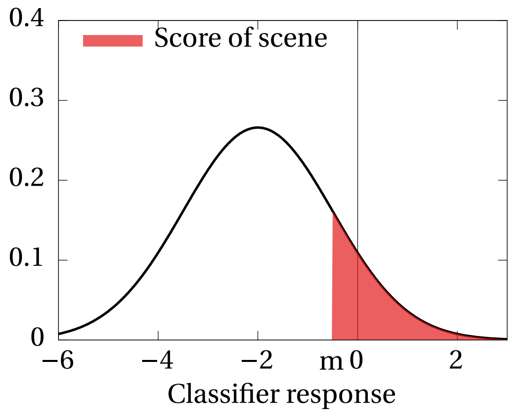


Figure 3.5 – Erf score for a scene

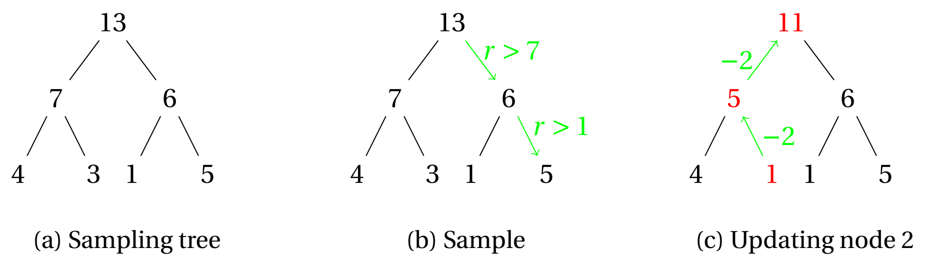


Figure 3.6 – Sampling according to a changing discrete distribution. Given a binary tree built on top of a discrete distribution $[4,3,1,5]/13$ (figure 3.6a), we can draw a sample according to this distribution by recursively sampling according to the partial distribution of the children at a given node (3.6b). Only the nodes between the leaf to update and the root need to be updated (3.6c). Both operations have $\mathcal{O}(\log N)$ complexity.

All sampling methods except the uniform strategy require to sample according to positive weights (image size, unexplored area, or erf score) normalised into a discrete distribution that may evolve in time (for unexplored area and erf score).

Sampling among N items using a (balanced) binary tree costs $\mathcal{O}(\log N)$ both in sampling and updating the distribution. Figure 3.6 shows an example for the following distribution $[4,3,1,5]/13$. Given the distribution (in our case, it represents the scores of the scenes), a binary tree is build such that the leaves contain the scores of the images and each internal node have the sum of both its children (see 3.6a). To sample according to this distribution, one starts from the root and recursively draws a number r between 0 and the value of the current node: if r is smaller than the left child, it restarts the sampling with the left child. Otherwise it chooses the right child (see 3.6b). The process goes on until reaching a leaf. To update a particular leaf, one just has to update all the nodes from the leaf up to the root (see 3.6c), which also costs $\mathcal{O}(\log N)$.

3.3 Experiments

We now apply our strategy to train a pedestrian and a face detector. We compare how many false positives are actually found with respect to the method that does not discard a neighbourhood, and we compare the strategies to select scenes. A hard sample is defined with $m = 0$.

The computational cost of bootstrapping is proportional to the number of visited locations since the time to choose a scene (implemented with a binary tree), then a position, then to add the neighbourhood to V_n is negligible in comparison to computing $f_1(x)$. Hence, we report the results as functions of the number of visited locations. The positive discarding radius r^+ is set to 0 for the faces and to 2 for pedestrians.

3.3.1 Data sets

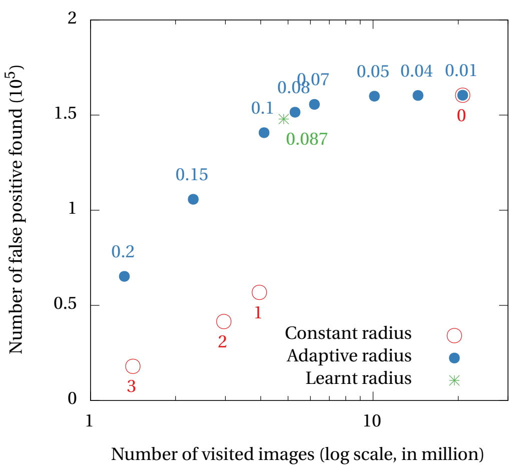
We use the INRIA pedestrian data set (Dalal and Triggs, 2005) to train the pedestrian detector. It consists of 2,416 128×64 images of pedestrians and 1,218 pedestrians-free scenes for training. The testing set consists of 1,132 pedestrians and 453 scenes. We train the initial detector by using 10,000 128×64 negative images randomly extracted from the training scenes. We use HOG as features and we use the SVM library LIBLINEAR (Fan et al., 2008) with a ℓ_2-regularised ℓ_2-loss.

For faces, we use the standard face data set of Viola and Jones for training and test it on the MIT-CMU data set. As negative scenes, we used the Caltech Background data set (Weber) which provides 900 face-free images (we removed 3 images containing a skull or a baby). We use AdaBoost (Freund and Schapire, 1997) as the learning algorithm and Haar-like features (Viola and Jones, 2004).

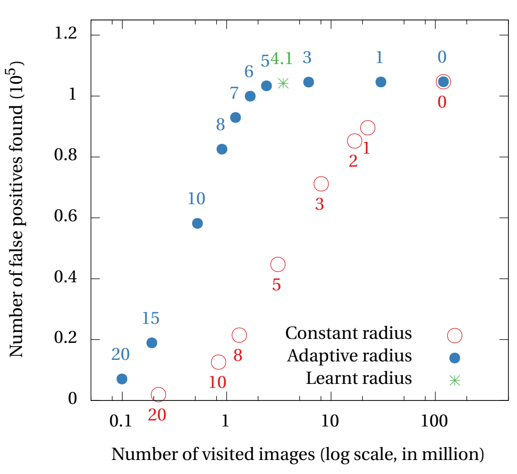
3.3.2 Number of hard samples found

Although our strategy is designed to discard the largest expected neighbourhood that does not contain any false positives, some of them might actually be in this discarded area. To analyse this, we compare our *learnt radius* as described in 3.2.1 with a manually chosen radius, *i.e.* using $a \gg a^\star$ (resp. $a \ll a^\star$), that is with a more aggressive (resp. smoother) slope. We therefore compare our strategy to *exhaust* all the scenes with:

- Parsing densely all the scenes *without* discarding any neighbourhood (*i.e.* discarding with a radius $r^- = 0$) (red circle with label 0 on figure 3.7).
- Densely parsing all the scenes, and discarding a constant radius around true negatives, *i.e.* disregarding $f_1(x)$ (red circles on figure 3.7).
- Densely parsing all the scenes, and discarding a manually selected adaptive radius, *i.e.* $r^- = -af_1(x), a \neq a^\star$ (blue dots on figure 3.7)



(a) MITCMU data set



(b) INRIA data set

Figure 3.7 – Number of false positives found vs. the number of visited samples for the different strategies (average over 10 runs). In the experiments, we visite the scenes until *all* are exhausted. The red circles correspond to a strategy which discards a disc of constant radius around a visited sample x, the blue dots to the strategy which uses an adaptive radius $-af_1(x)$. The learnt radius (green star) is in the area where almost all the available false positives are found while visiting a minimum of positions.

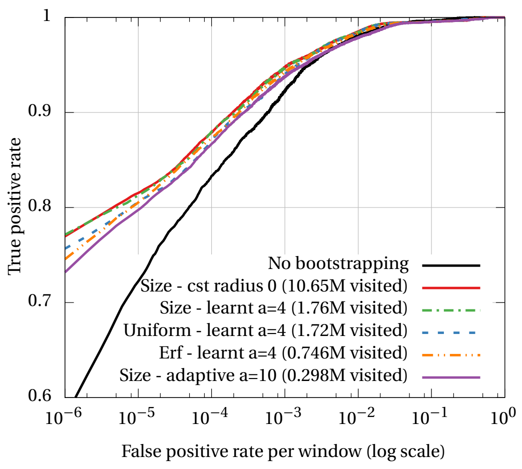
Figure 3.7 shows the results of the experiment: we plot the number of false positives found as a function of the number of visited locations across all the scenes (recall that we exhaust all scenes).

The plateau formed by the blue dots corresponds to the maximum number of false positives we can actually find in the scenes. With a constant radius (red circles) the parsing discards the false positives which are neighbours of true negatives, and as a result, finds less hard samples than an adaptive radius.

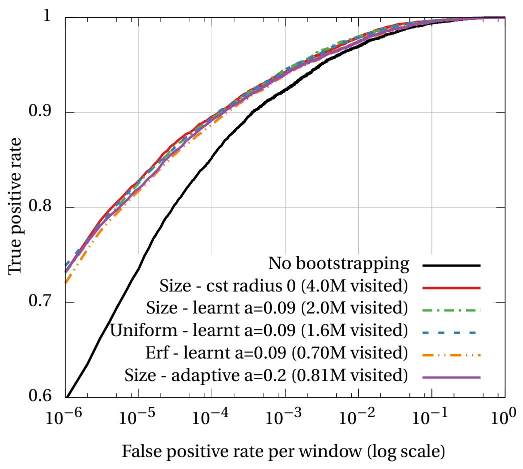
An adaptive radius does not discard false positives next to negative samples because we have

$$r^- = -af_1(x) < 1.$$

For both data sets, our simple method finds a learnt adaptive radius which lies in the area where one visits the fewer positions, while finding almost all the available hard samples. We argue that any model or estimation method that leads to such ratio between number found and number visited is sufficient to have a significant reduction in the number of visited locations. In this experiment, we visit 30 times (resp. 4) fewer locations on the INRIA data set (resp. Caltech background).



(a) Pedestrian detector on the INRIA Person set



(b) Face detector on the MITCMU data set

Figure 3.8 – Comparison of the performances on the two test data sets for different sampling and discarding strategies (averaged over 20 runs). The initial detector (black curve) is trained as described in section 3.3.1. Choosing a scene according to its size and using the learnt adaptive radius yields better performances than uniform, erf or a more aggressive slope. It also visits 6 times (resp. 2) less samples than the traditional approach and reaches similar performances. We note that for faces, the selection strategy does not matter.

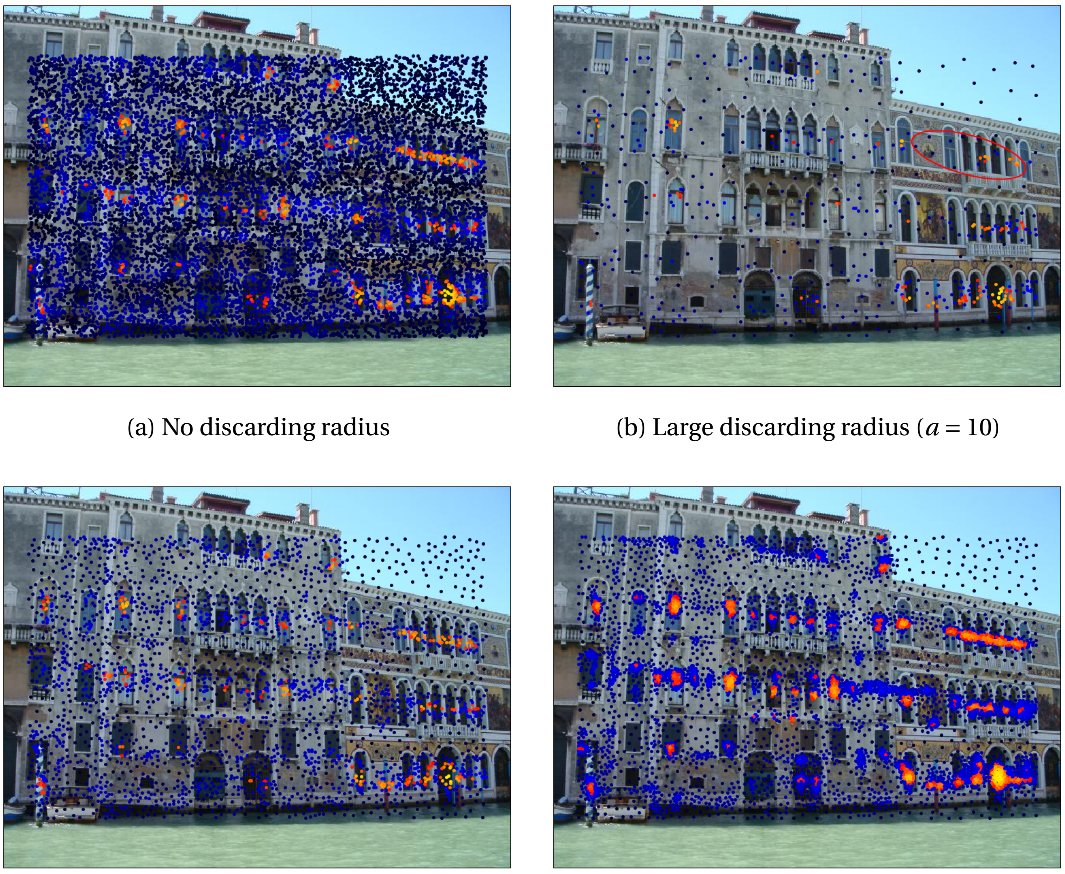
3.3.3 Performances

Our ultimate objective is to produce the best detector we can. While the number of samples used for training is an important parameter, it is not the only one. Diversity, and more generally the “representativity” of the sample set, is key. An extreme example would be a very large set composed of the same example replicated with very minor pixel noise variation.

Regarding the collection of samples from scenes, if we were only interested in the sheer number of hard samples we collect, we could simply mine exhaustively in the neighbourhood of any hard sample we find, since it is often the case that hard samples come in dense clusters.

To assess the efficiency of our method with this aspect taken into account, we must estimate not only the computational cost to mine K false positives, but also how good they are for training. To that purpose, we compare the various strategies after one round of bootstrapping adding $K = 30,000$ hard samples (figure 3.8). Figure 3.8a presents the performance of the pedestrian detector and figure 3.9 shows the visited locations in a particular scene for various methods.

The red dashed curve is the baseline which does not discard any neighbourhood ($r^- = 0$). It needs to sample 10 million positions to find 30,000 hard samples. As we see on figure 3.9a, lots of locations are visited. Our proposed strategy (green line) needs to sample 6 times fewer positions and yet achieves the same accuracy. We have not noticed much difference between sampling a scene according to its size or to its unexplored area so we report the results for



(a) No discarding radius

(b) Large discarding radius ($a = 10$)

(c) Learnt discarding radius ($a \simeq 4$)

(d) Erf sampling and learnt discarding radius

Figure 3.9 – Comparison of the visited locations for different strategies. One dot represents the top left corner of the extracted window $a = 10$ tends to discard too wide regions that causes some small areas of false positives to be discarded.

size only. We see on figure 3.9c that our method discards larger neighbourhoods around true negatives with large negative response f_1 (dark points) and concentrates its sampling on the areas where the false positives lie.

Our approach with uniform sampling performs slightly worse than the size or unexplored because it unbalances the sampling: small scenes have equal probability of being chosen so $\mathcal{H}$ contains more samples from small scenes.

Too aggressive a slope ($a = 10$ here whereas our learnt slope is $\simeq 4$) performs also worse. Discarding too large areas tends to discard plenty of false positives as we can see on figure 3.9b. Some “small islands” of false positives were discarded (red circled area). Large “islands” of false positives then tend to be over-sampled that yields a set of redundant samples, as mentioned at the beginning of 3.3.3.

Erf sampling performs also worse because it tends to select the same scenes so $\mathcal{H}$ is unbalanced

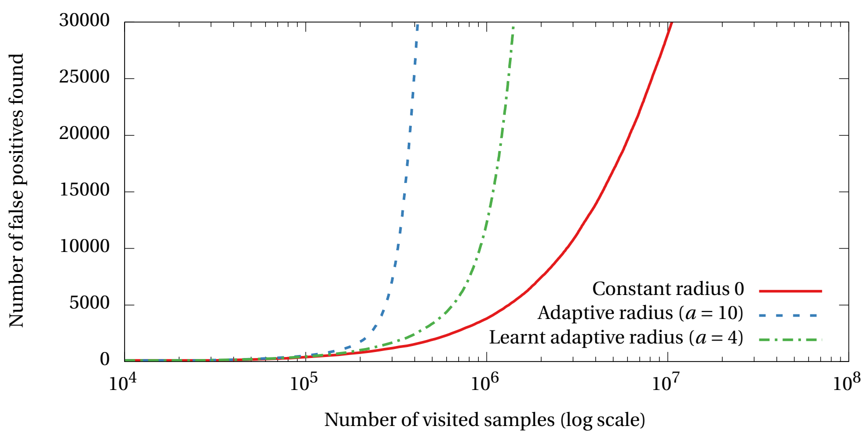


Figure 3.10 – Number of false positives found as a function of the number of visited locations. The curves show that in the beginning of the procedure, all three methods find the same number of false positive samples but that after a certain point, the adaptive radii find more. The process has discarded lots of areas and only the false positives remain.

(on 3.9d the scene is more visited than if it is sampled according to its size). Without a bound on the erf score, *i.e.* without $1/\pi r^{+2}$, the performance dramatically drops because the hard samples come from very few scenes.

Finding hard samples goes much faster as we visit more locations because more areas have been discarded in the early steps as depicted on figure 3.10, where the number of false positives found increases exponentially. On figure 3.10, we targeted 30,000 false positives, and the speedup of the learnt adaptive radius is 6 fold over the constant radius. On figure 3.7b, the speedup was 30 fold because we exhausted all the scenes.

The right plot of figure 3.8 presents the performance of the face detector. All strategies give the same performance but an adaptive radius reduces considerably the number of visited locations and thus the training time.

3.4 Discussion

The simple strategy presented in this chapter aims at mining scenes to find hard samples for object detection. It consists in learning an adaptive radius to discard a neighbourhood around a true negative sample as a function of the initial classifier response to avoid visiting these discarded locations in further steps of the parsing. Selecting a scene according to its size has also showed to yield better results.

This strategy allows to find as many hard samples as the naive approach which visits all possible locations an order of magnitude faster, while achieving similar performances.

The work of Yan et al. (2014) have proposed another approach which is somewhat reminiscent of the one described in this chapter. Their “neighbourhood aware cascade” is based on the same observation as ours, which is the dependency between scores of neighbour positions. They derived a pruning strategy of neighbour positions by using a threshold depending on the stage of the cascade. In addition to that, they extended the pruning to positive scores, as they observe that on average, before non-maximum suppression, a detection comes with 21 other positive detections. In our case, we would not use that because we are especially interested in finding all false positives.

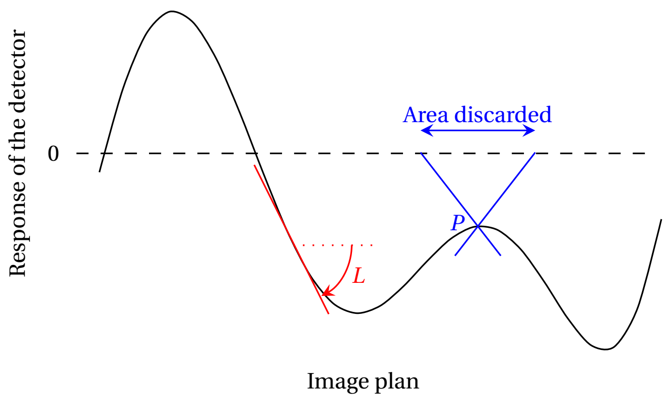


Figure 3.11 – Interpretation of the method with a Lipschitz constant

Another way of interpreting the method presented in this chapter is to consider the response of the detector in the image plane as a 2D Lipschitz continuous function, whose Lipschitz constant is actually the slope of lower bound depicted on figure 3.4. When the classifier is evaluated at a position P and has a negative response, given the Lipschitz constant L which is the largest rate at which the function varies, we can compute the distance from P around which the function cannot change sign. Therefore the area discarded corresponds to the largest area around which the function cannot go above 0. This interpretation is depicted on figure 3.11 in 1D: the curve represents the response of the classifier as a function of one coordinate in the image plane, and the red line is the steepest tangent of the curve, which slope is therefore equal to the Lipschitz constant.

4 Monte Carlo Tree Search Bootstrapping

Contents

This chapter describes a new framework to efficiently find informative images in a large collection of images to train object detectors with the hard negative mining approach. The main idea is to organise the images in a tree structure that reflects the regularities of the collection and to use a tree search approach to choose the images. This chapter is based on the following publication:

Olivier Canévet and François Fleuret. Large scale hard sample mining with Monte Carlo Tree Search. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016

4.1 Introduction

As we have described in section 2.2.1, training a sliding-window object detector requires a vast amount of negative images for the bootstrapping procedure. These object-free images are source of hard negative samples, which are added to the training set to improve the boundaries between the population of positive and negative samples.

4.1.1 Statistics of hard samples within images

In practice, one observes that the frequency of hard samples in images is highly structured: certain types of images exhibit statistical regularities that generate more or less frequent hard samples (figure 4.1). Similar structures can be observed in the images themselves: large uniform patches (sky, empty walls) can be ignored (figure 4.13a), while high-frequency or highly-structured parts (trees, buildings, bookshelves) should be examined in detail (figure 4.13b). If one has to collect images from the web to create a "good" set of background images, (s)he would quickly get a good intuition about which images to select and which to ignore.

4.1.2 Structure of image data sets

Moreover, one also observes that the image collections have an inherent structure, which can mostly be accounted for by how the images were originally collected. For instance, the PASCAL (Everingham et al., 2015) and COCO (Lin et al., 2014) data sets contains many images of planes in a blue sky, landing planes, vehicles in the same position, and animals. And related to the observations of the previous section, similar images tend to generate the same amount of hard samples. Figure 4.2 shows examples of similar images in the content they depict (sky, desert, animal, flower) and a histogram of the number of false positives that can be found in such images. Images of sky and desert happen to be poor in hard samples (many images have less than 5 false detections), while images of animals and flowers are richer: few of them do not contain false positives, and many have between 5 and 15 false detections.

The quality of images as sources of hard samples is therefore strongly related to the geographical environment or type of events they depict, or indirectly to the time period, photographer, or even the web site they originate from. In a video for instance, time-consistency induces a strong regularity of the proportion of hard samples in contiguous frames.

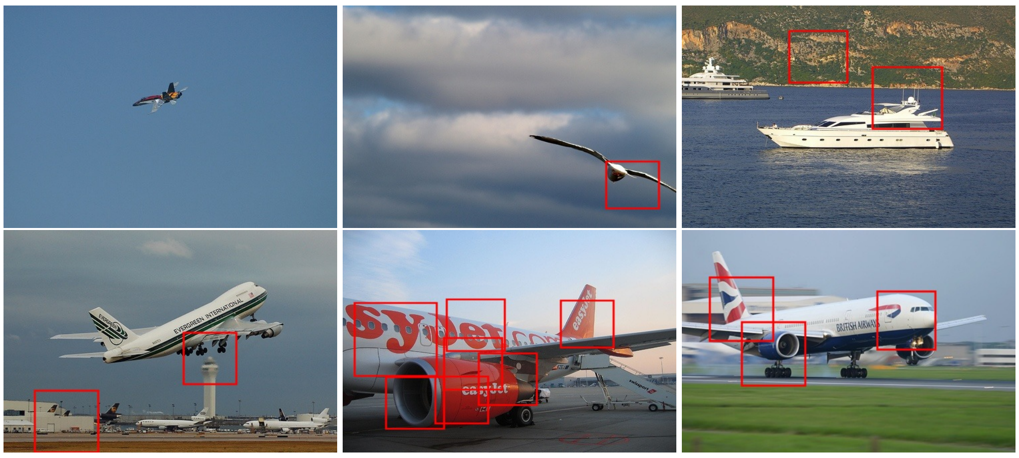
4.1.3 Motivation

The existence of such structures motivates the use of a hierarchical process able to concentrate computation recursively, figuring out automatically at what scale (image sets, image subsets, image) it should make a decision about investing or not more computation in the corresponding samples.

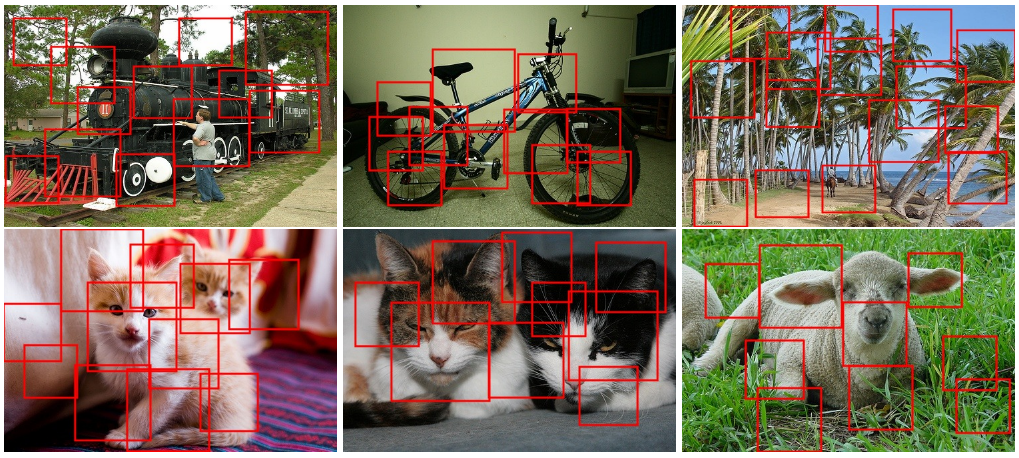
We propose to formalise the problem by first defining a tree-structure whose leaves are individual images, and whose nodes correspond to small groups of content/temporal related images in the bottom level (street, flowers, indoor, etc.), and larger groups of data set related images in the top level (data set, origin, etc., see figure 4.6 for an example). If the structure is given (temporal structure, keywords, etc.) then no preprocessing is needed. Given such a tree and an existing predictor, each leaf (*i.e.* image) is labelled with a score that reflects how many false positives it contains.

Our objective is to use that tree-structure to efficiently sample among false positives, that is to maximise the fraction of false positives we find among the samples we actually look at.

Without an additional structure, this problem amounts to an exploration-exploitation dilemma:

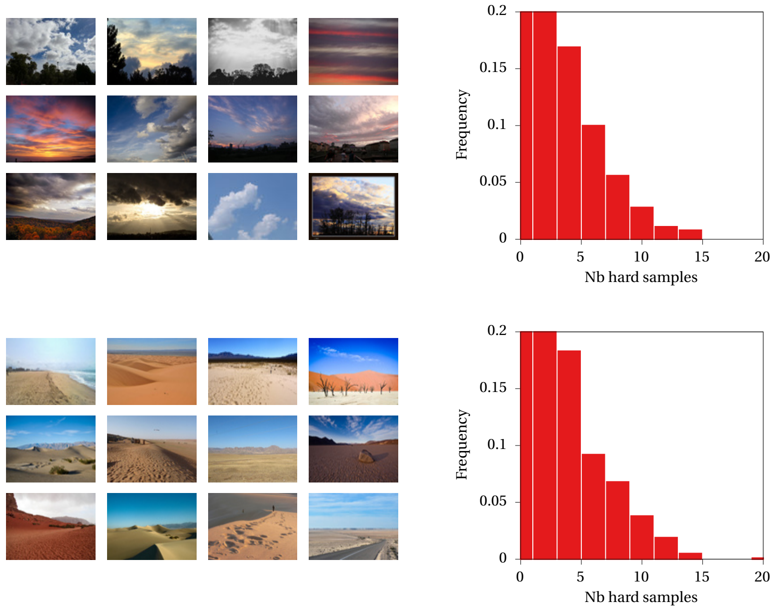


(a) Images poor in hard samples

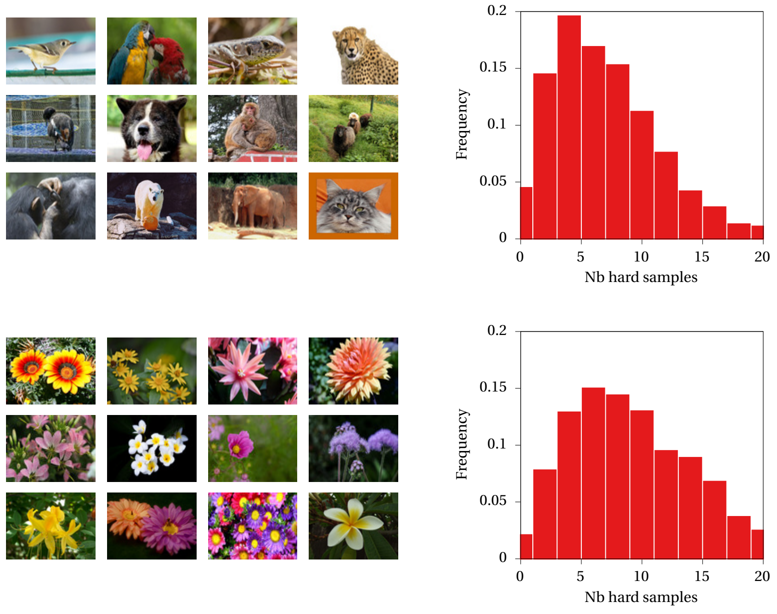


(b) Images rich in hard samples

Figure 4.1 – Frequency of hard samples in background images. An aggregated channel feature face detector trained with 128 trees is used to collect hard samples in object-free scenes. The detections (red squares) are false detections.



(a) Images poor in hard samples



(b) Images rich in hard samples

Figure 4.2 – Frequency of hard samples in various type of images. Images of skies and deserts are poor in hard samples, many of them have less than 5 false positives, while images of animals and flowers are richer, and many of them have between 5 and 15 false positives.

we want at the same time to “exploit” the groups of images we have already identified as promising, that is are rich in false-positives, but we also want to invest a fraction of our computational effort to “explore” new groups of images.

Framed in such a way, a natural response to the problem is the use of the Monte Carlo tree search (MCTS). This technique associates a multi-armed bandit to each node of the tree, and uses these bandits to sample paths down the tree, based on the current estimates of rewards, or in our case, on the proportion of hard samples in the sub-trees. While MCTS is traditionally used to estimate the good choice to make at the top node, the by-product we use here is the list of leaves it has visited during sampling.

4.2 Background on Bandit methods and Tree Search

This section presents some background on the bandit problem and how it has been ported to tree structures in Monte Carlo tree search. These strategies belong to the field of reinforcement learning, which deals with how a machine can learn the optimal behaviour from its environment. The machine interacts with its environment by receiving a reward after each action, and its goal is to maximise its cumulative reward over the long term by choosing the most profitable action. Reinforcement learning is therefore concerned with the exploitation-exploration trade-off, because the machine should select the empirically best action observed so far, but at the same time, needs to invest a fraction of its computational effort on exploring other actions.

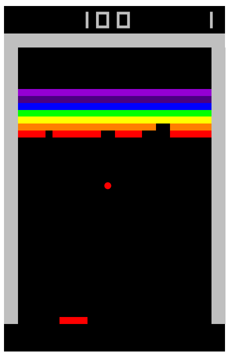


Figure 4.3 – Screenshot of the Atari Breakout game

A practical example of reinforcement learning is training a computer how to play a game simply by having access to screenshots of the game. The main motivation is that a human would be able to understand the rules of the game and how to play by simply looking at the course of a play of another player. We illustrate this scheme with the Atari Breakout game (figure 4.3) that was used by Mnih et al. (2015). The goal is to let the machine find an optimal policy to play this game. The machine only knows that it can move the platform left or right, which constitutes its set of possible actions, and it has only access to the screenshots of the game (10 images per second) and to the current score. Whether the score increases based on the chosen actions is the reward, and the machine will learn how to maximise this score while playing.

4.2.1 The Multi-Armed Bandit problem

The multi-armed bandit problem introduced by Robbins (1952) is a statistical model of the decisions an agent should make to properly balance between exploration and exploitation. The traditional comparison used to present the multi-armed bandit problem is the one of a gambler in a casino playing with K slot machines (*i.e.* each of them known as one armed bandit, see figure 4.4) and the process is the following: the gambler (*i.e.* the agent) selects one out of K arms, inserts a token, pulls the arm (*i.e.* the action) and possibly receives a reward (*i.e.* the feedback from the environment).

The main objective of the player is to maximise his/her cumulative reward, or equivalently to minimise his/her cumulative regret, which is the loss due to not playing the optimal machine all the time. In its stochastic version, a bandit is described by a fixed probability distribution of parameter θ_k, of support in $[0,1]$ and of unknown expectation μ_k. Playing arm k generates a reward, which is drawn from the corresponding distribution, and is assumed to be independent of the previous draws.

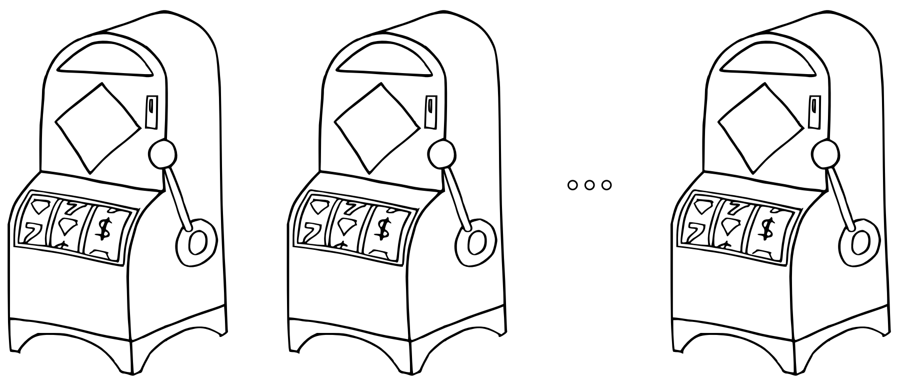


Figure 4.4 – Slot machines

We call $\mu^\star$ the expectation of the best arm,

$$\mu^\star = \max_k \mu_k, \tag{4.1}$$

and $\sigma(t)$ the selected arm at time t. Then the regret r_t at time t is

$$r_t = \mu^\star - \mu_{\sigma(t)} \tag{4.2}$$

and the cumulative regret R_n up to time n is

$$R_n = \sum_{t=1}^{n} r_t = \sum_{k=1}^{K} T_k(n)(\mu^\star - \mu_k), \tag{4.3}$$

where $T_k(n) = \sum_{t=1}^{n} \mathbb{1}[\sigma(t) = k]$ represents how many times arm k was pulled up to time n.

Lai and Robbins (1985) proved that the regret grows at least logarithmically with the number of plays and therefore, solving the multi-armed bandit problem consists in finding a policy to select the next arm to pull, given past observations and to achieve a logarithmic regret.

The multi-armed bandit problem finds its application in various fields. Originally, it was developed for medical trials, in which one wants to correctly identify the best treatment (exploration), and treat patients as effectively as possible during the trial (exploitation). It is currently used to efficiently select advertisement on the web, for which placing an ad corresponds to pulling an arm, and the user's click is the reward.

Upper Confidence Bound algorithm (UCB1)

One popular policy solving the multi-armed bandit problem is the upper confidence bound algorithm proposed by Auer et al. (2002) and asymptotically achieves a logarithmic regret. UCB1 is summarised by algorithm 1. The player knows how many times n_k (s)he has already played arm k and the total reward w_k (s)he got from that arm. We call $X_k = \frac{w_k}{n_k}$ the empirical expectation of arm k. At time t, the player pulls the arms that maximises:

$$X_k + \sqrt{\frac{2 \ln t}{n_k}}. \tag{4.4}$$

After playing arm k, n_k is incremented and w_k is updated accordingly.

Equation 4.4 consists of two terms. The first one, $X_k = \frac{w_k}{n_k}$, is the expectation of arm k and corresponds to the exploitation part of the score: an arm yielding more rewards will have a higher expectation. The second one, $\sqrt{\frac{2 \ln t}{n_k}}$, is the exploration term: a frequently visited arm will have a lower exploration part (n_k higher) while a less frequently visited arm will have a larger exploration score. However, as t increases, the logarithmic term will be dominated by the linear part (the denominator), and asymptotically, only the best arm(s) are pulled.

Algorithm 1 Upper Confidence Bound algorithm (Auer et al., 2002)

```
$\forall k,\ X_k \leftarrow 0, n_k \leftarrow 0$
for $t = 1$ to $T$ do
    Select arm $k_t = \text{argmax}_k X_k + \sqrt{\frac{2 \ln t}{n_k}}$
    Observe reward $w_t$ from arm $k_t$
    Update $X_{k_t}$ and $n_{k_t}$
end for
```

Thompson Sampling

Thompson sampling was introduced by Thompson (1933) in order to address the exploration-exploitation trade-off in a purely Bayesian manner. It was recently applied for the multi-armed bandit problem by Chapelle and Li (2011), and then proved to achieve a logarithmic regret (Agrawal and Goyal, 2011). The idea of Thompson sampling is to assume a prior distribution on the parameter θ_k of the distribution of each arm, and at each iteration t, to play the arm according to its posterior probability of being optimal. At time t, the sampling selects the arm maximising

$$\mathbb{E}[X_k|\theta_k], \tag{4.5}$$

where $X_k = \frac{w_k}{n_k}$ is the expectation of arm k, and θ_k is drawn from the posterior at each iteration. Here, w_k depends on θ_k. The workflow of Thompson sampling is described by algorithm 2.

Algorithm 2 Thompson sampling (Chapelle and Li, 2011)

```
1: $D \leftarrow \emptyset$
2: for $t = 1$ to $T$ do
3:     For each arm, draw $\theta_k \sim P(\theta|D) \propto P(D|\theta)P(\theta)$
4:     Select arm $k_t = \text{argmax}_k \mathbb{E}[X_k \mid \theta_k]$
5:     Observe reward $w_t$ from arm $k_t$
6:     $D \leftarrow D \cup (k_t, w_t)$
7: end for
```

The use of Thompson sampling is motivated by the fact that it can explicitly embed a model of the rewards with a long tail distribution as opposed to UCB1 which are constrained to be in the full range $[0,1]$.

In this setup, the exploration phase occurs in the beginning of the sequence when the posterior distributions are not estimated with many observations, and as the number of observations increases, the estimation of the posterior is better, and as for UCB1, only the best arm is pulled asymptotically.

4.2.2 Monte Carlo Tree Search

Monte Carlo tree search (MCTS, Browne et al. 2012) is a method to find the optimal solution in a given and potentially huge search space. MCTS balances between analysing promising moves in the space (exploitation) and expanding the tree randomly (exploration). It has gained a lot of interest in the artificial intelligence community in the last decade because of the huge improvement it brought in the game of Computer Go.

MCTS has successfully been applied to games to make a computer play against a human. Previous strategies such as $\alpha\beta$ (Knuth and Moore, 1976) or A^* (Klein and Manning, 2003) have shown to be efficient against humans for the games of chess and checker, because they have a low branching factor (the number of possible moves to make) and because it is quite easy to evaluate the outcome of the game given the current state. But for games such as Go, computers have long been unable to defeat non-professional players until MCTS was introduced. By doing randomised simulations of the game and biasing simulations towards a successful end for the computer, MCTS is able to find the next best move to be made. Many works have been done to formulate MCTS for games such as the ones by Coulom (2007), Kocsis and Szepesvári (2006), or Chaslot et al. (2006) and computers (AlphaGo) are now able to defeat professional players on the traditional board of Go (Silver et al., 2016). Other works have been focusing on speeding up the exploration to get a better estimate of the best move to make (Bourki et al., 2011) or incorporating expert knowledge (Chaslot et al., 2010).

MCTS has also been used in video games to simulate an intelligent behaviour against the human in real-time. If for board games the computer (and also the human) have several minutes to make a decision, video games are real-time and require making a decision within a few seconds. MCTS is well suited for this purpose because it balances efficiently between exploring the space of decisions and in the end avoids bad decisions. It was recently used in the game Total War: Rome II (Champandard, 2014) whose purpose is to build a civilisation in the Antiquity period, develop it and conquer other civilisations. When a human player plays against the computer, the latter uses MCTS to make good decisions, allocate resources and task efficiently in the civilisation.

Another example of successful use of MCTS is the optimisation of a “black-box” function (Munos, 2014; Coquelin and Munos, 2007) where the goal is to get a good estimate of the maximum of the function (deterministic or stochastic) by evaluating it only a limited number of times. The idea is to design a sequence of input samples on which the function should be evaluated given the previously observed values. More precisely, starting from the entire input space, the method recursively partitions it in subspaces, in which the function is evaluated. The space is split in a hierarchical manner, and MCTS determines in which subspace the function should be evaluated next. As the process goes on, the procedure converges to the subspace where the function is maximal.

In all these applications, the input space can be represented with a tree that serves as the support for MCTS. To give more details on the basic run of MCTS, we use the case of a two-player game, in which the machine tries to determine the most promising move to be made against the human player.

For every move, one considers a tree whose root node corresponds to the current configuration of the game, whose internal nodes correspond to possible future configurations, and whose leaves are end-of-game configurations. Using this tree, the sampling procedure of MCTS (see figure 4.5) recursively goes down the tree as follows: in every node, if some of the children have never been visited, one is selected at random uniformly. If all children have been visited at least once, the selection is framed as a multi-armed bandit problem (Auer et al. 2002 and described in section 4.2.1) to optimally tackle the exploitation-exploration dilemma. When a leaf is reached, that is a wining configuration for one of the two players, the reward is backpropagated up to the root, and the statistics at each node regarding the number of times it was visited and the fraction of winning outcomes are updated.

This sampling is repeated until a computational, or time budget is exhausted, at which point

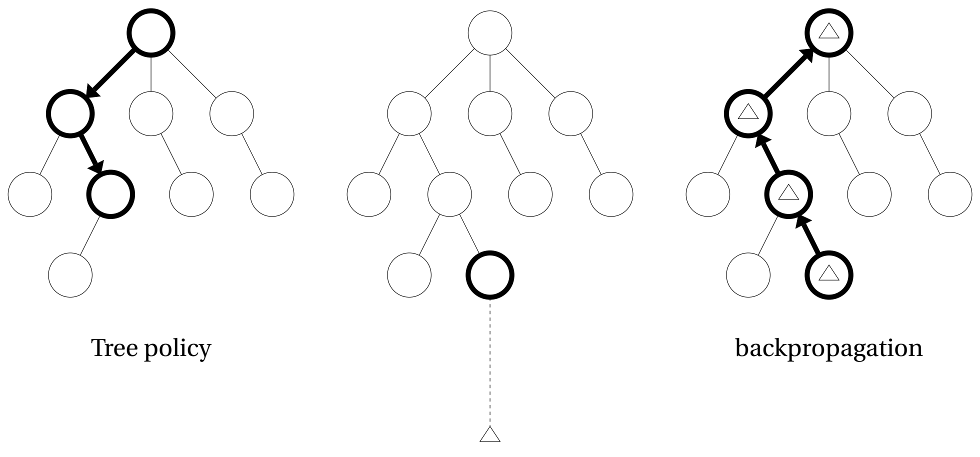


Figure 4.5 – One run of the traditional MCTS (reproduced from Browne et al. 2012)

the next move is made by selecting the best child of the root which is the one with the maximum proportion of wins.

MCTS and bandit algorithms have nice theoretical properties, in particular, the guarantee that they only expand the optimal part of the tree. Moreover the tree structure of MCTS allows to deal with very large spaces leaving unexpanded unpromising parts of the domain.

As explained in detail in section 4.3, we propose to formulate the problem of mining hard samples in a MCTS way. We associate an image to each leaf, and a positive reward if it contains false positives. However, instead of using the MCTS to eventually select a good child at the root node, we keep track of the "good" samples to retrain the classifier.

4.3 MCTS Bootstrapping

4.3.1 Image data set structure

As said in section 4.1.2, image data sets inherently have a hierarchical structure by the way images were collected, and our procedure builds upon this structure (see figure 4.6).

In the top level, each child of the root corresponds to a specific image data set, such as PASCAL (Everingham et al., 2015), COCO (Lin et al., 2014), or any image directory available on one's hard disk.

Further down the tree, nodes correspond to subsets of data sets, such as the year for PASCAL (2007, 2008, etc.) or the name of the semantic object contained in the sub-directory for Flickr (desert, animal, etc.). Finally, at the bottom of the tree, leaves are individual images.

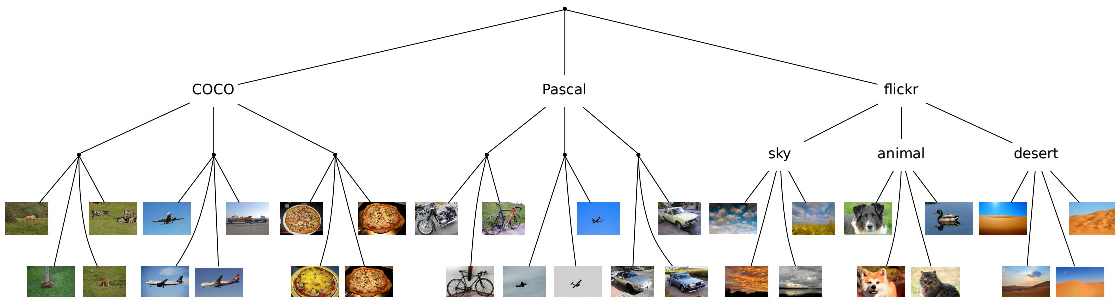


Figure 4.6 – Example of a structured image database. The structure can be explicit such as in the Flickr data set (sky, animal, etc.) or implicit like in the PASCAL data set, where images can be grouped either based on their names or after a pre-clustering of the images. Videos have a temporal structure.

When an image data set comes with no explicit structure (such as Microsoft COCO or PASCAL), a clustering can be applied to build subgroups of visually similar images, which corresponds in practice to coarse semantic categories of similar structural complexity.

4.3.2 Procedure

We now explain in detail how MCTS Bootstrapping works to train an object detector. We assume we have this very large structured database of object-free images.

After training the initial detector with the collection of positive samples and a collection of negative samples uniformly taken in the data set, the detector is bootstrapped several times by adding false detections.

We recall that in the traditional setting, the detector is applied on random images from the data set until finding enough hard samples.

In MCTS Bootstrapping, the next image on which to apply the detector is chosen by traversing the tree from the root in an MCTS fashion. A first multi-armed bandit problem selects the data set from which the image will be chosen. Then a second multi-armed bandit selects from which subpart of this data set, etc., until eventually reaching an image. The detector is applied on it, the hard samples (if any) are kept (that is receiving a reward) and the outcome of the play is eventually backpropagated up to the root by updating the various statistics of all the nodes that were traversed. The image is marked as "exhausted" not to be selected anymore in the future steps.

This process is then repeated to select another image, this time based on the new updated statistics, until enough hard samples are found. As hard samples are found, the MCTS policy progressively concentrates its sampling on more promising parts of the tree, hence data sets, that is on subgroups of images which are rich in hard samples.

Statistics are reset to 0 before beginning a new bootstrapping phase because images that produced false positives in previous rounds may no longer be informative, and also because the detector has changed after retraining.

4.3.3 Scores

As described in section 4.3.2, each node contains the number of wins that were obtained after traversing it, and the number of times it was traversed. We propose to adapt these scores to our scheme of MCTS Bootstrapping. We also recall that the rewards should be in the *full* range $[0,1]$ (Auer et al., 2002).

As the goal is to find false detections, the reward obtained after applying the detector should be a function of the number of hard samples h in the image which was eventually selected. The first score that we can define is

$$\text{win} = \begin{cases} 1 & \text{if } h > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

This "win" score corresponds to the vanilla setup of the multi-armed bandit where the reward is either 0 or 1 ("did the player win or not?"). The backpropagation rule to update the score of node p with the scores of its children $\mathcal{C}(p)$ is

$$\text{win}_p = \sum_{c \in \mathcal{C}(p)} \text{win}_c. \tag{4.7}$$

However, the score should reflect the size of an image, to leverage the fact that finding the same amount of false detections in two images of different sizes does not require the same (computational) cost. A natural score would be

$$d = \frac{h}{S}, \tag{4.8}$$

where S is the number of times the detector is evaluated in the image, which is proportional to its size in pixels. d is thus the *true* density of false positives in the image.

Preliminary experiments show that the *true* density does not suit UCB1 policy because the rewards are small. A detector can be evaluated ~100,000 times on a 640×480 image so finding 10 hard samples leads to a reward of 0.0001. The exploration term of equation 4.4 dominates the small exploitation score, as a result of which there is no exploitation.

We thus normalise this score so that it ranges in $[0,1]$ and finally, we define the *normalised* density

$$\tilde{d} = \min\left\{1, \frac{1}{2Z}\frac{h}{S}\right\}, \tag{4.9}$$

where Z is the mean of the density of hard samples that is estimated as the images are visited. The min ensures that the score lies in $[0,1]$. Basically, when the density of hard samples is around the mean, the reward is 0.5, and when an image is rich in false positives (*i.e.* h/S much larger than Z), the score is close to 1. The score is backpropagated the following way:

$$\tilde{d}_p = \sum_{c \in \mathcal{C}(p)} \tilde{d}_c. \tag{4.10}$$

So each node i contains the number of times n_i it was visited (0 or 1 for an image), the number of wins win_i (0 or 1 for an image), the number of hard samples h_i found below that node, and the number of times S_i the detector was evaluated. In addition to that, a Boolean flag indicates if there are still non-visited images below a node, thus avoiding going to "exhausted" branches.

Given the various scores, we define three different policies for the multi-armed bandit: two of them for UCB1 and one for Thompson sampling.

4.3.4 MCTS strategies

MCTS-UCB1-win (win) The first strategy is based on UCB1 (section 4.2.1) with the "win" score. At a given node p, the next node to select among its children $\mathcal{C}(p)$ is the one maximising

$$\frac{\text{win}_c}{n_c} + \sqrt{\frac{2\ln p}{n_c}}. \tag{4.11}$$

win_c / n_c is the average number of images in which at least one hard sample was found. This score reflects the probability of finding at least one hard sample in an image and corresponds to the simplest bandit scenario in which rewards are either 0 or 1.

MCTS-UCB1-dense (dense) The second one is based on the *normalised* density of hard samples and the next child to be selected is the node maximising

$$\frac{\tilde{d}_c}{n_c} + \sqrt{\frac{2\ln p}{n_c}}, \tag{4.12}$$

where $\tilde{d}_c$ is the *normalised* density of false positives as described in 4.3.3.

In both UCB1 policies, the exploration score $\sqrt{2\ln p / n_c}$ simply reflects time, just as in vanilla MCTS.

MCTS-Thompson-sampling (ts) For Thompson sampling (section 4.2.1) as described by Chapelle and Li (2011), a model of the distribution of the observations is required as well as a prior on its parameter. As previously stated, Thompson sampling does not require the rewards to lie in $[0,1]$ so we here directly use the *true* density.

Some simple experiments (see histograms of figure 4.2) show that the distribution of the density of false positives in images has an exponential shape. We thus model the *true* density d_i of hard samples with an exponential distribution

$$p(D|\lambda) = \lambda e^{-\lambda x} \quad \text{(with notations of algorithm 2, } \theta = \lambda\text{)}. \tag{4.13}$$

If we use the conjugate prior of the exponential distribution, that is the Gamma prior $\Gamma(\alpha, \beta)$, then the posterior is also a Gamma distribution with parameter $\alpha' = \alpha + n_i$ and $\beta' = \beta + d_i$, with n_i being the number of times node i was visited and $d_i = h_i / S_i$ the *true* density of hard samples.

So at a given node p with children $\mathcal{C}(p)$, the MCTS Bootstrapping based on Thompson sampling selects the next child by:

1. Drawing λ_c from $\Gamma(\alpha + n_c, \beta + d_c)$,
2. Selecting $\text{argmin}_c \lambda_c$ (the expectation of an exponential distribution of parameter λ is $1/\lambda$, so $\text{argmax}_c \mathbb{E}[X_c|\lambda_c] = \text{argmin}_c \lambda_c$).

We now compare these three strategies to the traditional uniform bootstrapping.

4.4 Experiments

We present the results of our experiments to train a face detector and a pedestrian detector with a large data set of images. We show that our MCTS bootstrapping approach is able to leverage the inherent hierarchical structure of the data set to efficiently find hard samples.

4.4.1 Detectors

We use the aggregated channel feature detector presented in section 2.2.3 which is one of the state-of-the-art detectors for pedestrian detection (Nam et al., 2014) and face detection (Mathias et al., 2014; Yang et al., 2014). We used in part the implementation[1] provided by the original author.

The deformable part-based model of (Felzenszwalb et al., 2010b) also reaches state-of-the-art performance for face detection (Mathias et al., 2014; Yan et al., 2014) and we used the Fourier-based implementation[2] of Dubout and Fleuret (2012).

[1] `https://github.com/pdollar/toolbox`

[2] `https://www.idiap.ch/scientific-research/resources/exact-acceleration-of-linear-object-detectors`

4.4.2 Image data sets with a tree structure

Caltech pedestrians

The pedestrian detector is trained on the Caltech pedestrian data set (Dollar et al., 2012) as described by Nam et al. (2014) by using 24,498 positive examples and 3 rounds of bootstrapping, each time adding 25,000 hard samples. The structure of the tree is straightforward because of the temporal structure of the data set. The sequences (set00-V000, set00-V001, etc.) are at the top of the tree while the images are arranged chronologically at the bottom of the tree.

Face-free images

The ACF (resp. DPM) face detector is trained with 15,000 (resp. 7,000) images of faces from AFLW (see figure 2.13 as examples) and we use a collection of 102,230 background (face-free) images. We have used 7,537 images from PASCAL and 24,685 from Microsoft COCO (Lin et al., 2014). In addition to that, we have downloaded images from Flickr using keywords which *a priori* are useful to train a face detector (animal, trees, etc.) or useless (sky, desert, landscape, etc.). On average, we collected 3,000 images of these categories.

The structure of the tree is obvious for "keyword" images: images of desert, trees or animals will each make a node (see figure 4.6). As for COCO and PASCAL, there is no inherited structure. To make one, we perform a pre-clustering of these data sets: we make a 48×64 thumbnail of each image, compute its features (gradient or GIST) and recursively perform a k-means clustering.

Figure 4.7 shows a small scale example of a top-down k-mean based clustering of the COCO data set. We see that a simple clustering such as this one is able to put similar images together: planes over blue sky, pictures of courses (pizzas, plates, food), landscapes with a strong horizontal line (sky on the top of the image, earth in the bottom), etc.

4.4.3 Results

MCTS Bootstrapping is faster than the traditional uniform approach because it is able to concentrate its search for hard samples on promising parts of the data set.

We present our results by looking at the number of images required to find the targeted number of hard samples. For the ACF detector, averaged computing times estimated on 40,000 images give 0.047s for computing the features and 0.011s for evaluating the detector per image. The Fourier-based DPM on 640×480 images requires 0.041s to compute the HOG features and 0.1s to do the convolutions. We therefore use the number of images as a comparison criterion, instead of the number of evaluations of the detector.

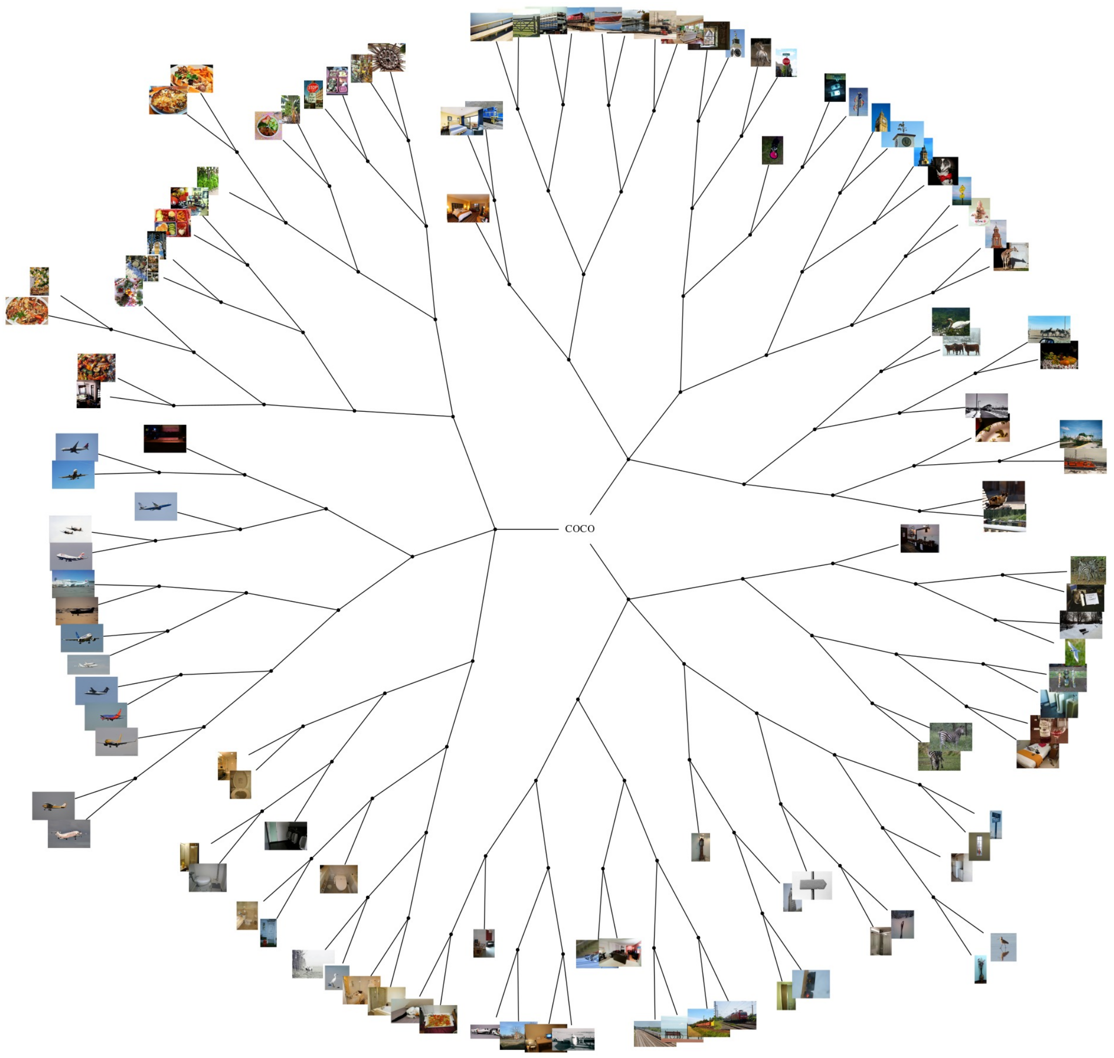
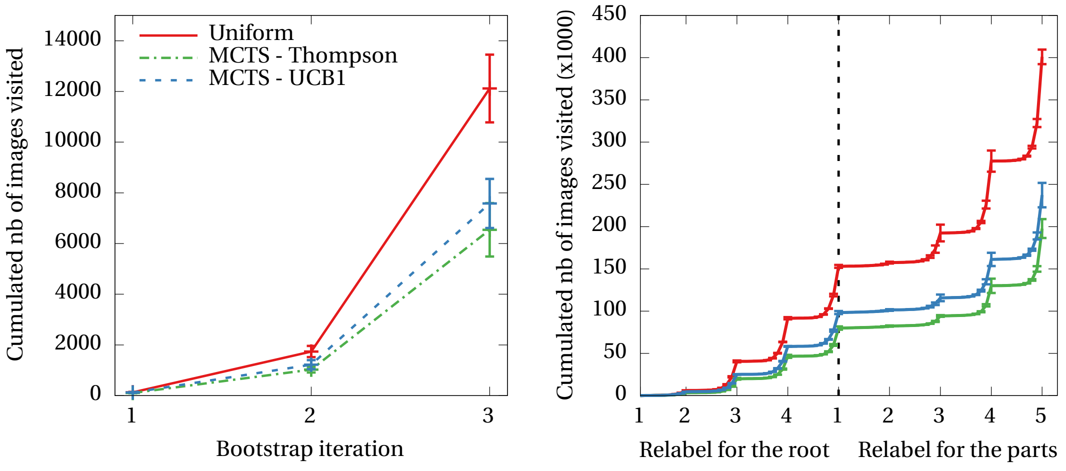


Figure 4.7 – A subset of Microsoft COCO data set organised as a tree with gradient-based top-down clustering. Similar images are put in the same branches. A MCTS-based approach can then identify which branches are promising to find hard samples after having selected a few images of the branches.



(a) ACF face detector (b) DPM face detector

Figure 4.8 – Number of images visited vs. bootstrapping iterations for a the ACF face detector and a DPM face detector. Our methods using a Monte-Carlo tree search (blue and green) focus on difficult images and visit roughly half the number of images the traditional approach needs (red).

Reduction of the number of visited images

Our MCTS-based methods need roughly half as many images than the traditional uniform approach, without hurting the performance of the resulting trained detector.

Table 4.1 presents the number of visited images required to train an ACF pedestrian detector on the Caltech pedestrian data set, as well as the final performance of the trained detector. Table 4.2 presents the same information in the case of an ACF face detector, and the cumulative number of visited images is depicted on figure 4.8a.

Regarding the DPM face detector, figure 4.8b show the cumulative number of images needed to train the root template and the parts of the model. Figure 4.9 presents the performance of this detector evaluated on the AFW and PASCAL data sets, and figure 4.10 shows the evolution of the performance as the training goes.

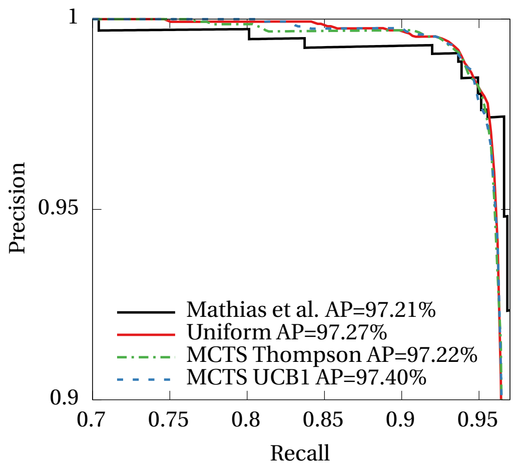
For both detectors (ACF and DPM) and for both types of object (face and pedestrians), MCTS Bootstrapping reaches the same performance as the uniform baseline, and in all cases, they require to parse half as many images as the baseline. MCTS is able to select images which are richer in hard samples, and which are also as informative as the ones found by the uniform approach.

	Nb images visited			Miss rate @ fppi		
Strategy	Boot1	Boot2	Boot3	0.01	0.1	1
uniform	568	2143	13767	0.46	0.29	0.16
win	563	2132	8885	0.46	0.30	0.16
dense	536	1843	7683	0.49	0.31	0.17
ts	509	1874	8625	0.48	0.31	0.17

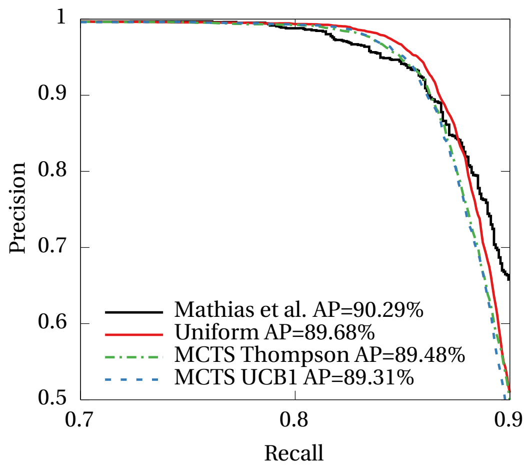
Table 4.1 – Number of images visited to train an ACF pedestrian detector with 3 bootstrapping steps (Boot1, Boot2, and Boot3, averaged over 10 runs) . The (video) data set is Caltech pedestrian and the temporal structure of the video was kept to build the tree. The performance on the Caltech pedestrian test set is the miss rate at a given number of false positive per image (fppi).

	Nb images visited			AP	
Strategy	Boot1	Boot2	Boot3	PASCAL	AFW
uniform*	130	1609	10377	84.4	95.3
win	126	1213	7811	84.0	95.4
win-grad	135	1164	5083	83.0	95.4
win-gist	128	1015	4828	83.8	95.1
win-shuf	132	1516	10035	84.0	95.4
dense	117	1271	7955	84.0	95.2
dense-grad*	122	1100	6359	84.2	95.5
dense-gist	115	880	5376	84.0	95.2
dense-shuf	137	1528	9931	84.1	95.6
ts	107	1172	6627	84.1	95.3
ts-grad*	106	919	5515	84.0	95.4
ts-gist	97	886	5123	83.8	95.3
ts-shuf	139	1496	9969	84.4	95.6

Table 4.2 – Number of images visited (normalised to 640×480) for each strategy to train an ACF face detector with 3 bootstrapping steps (Boot1, Boot2, and Boot3, averaged over 10 runs). The face images come from the AFLW face data set, and we used the collection of 102,230 images described in 4.4.2. The performance on PASCAL Faces and AFW is the average precision (AP), that is the area under the curves of figure 4.9. The *s indicate which strategies are also depicted in figure 4.8.

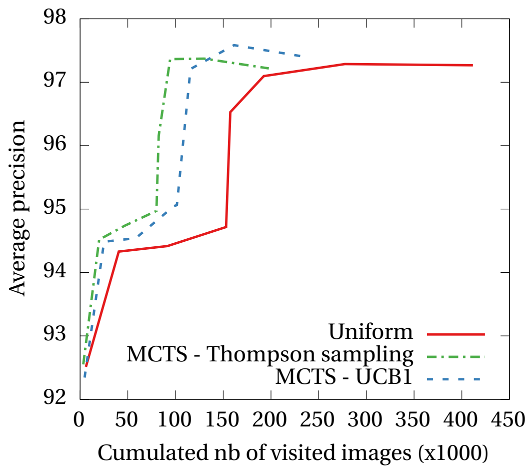


(a) Performance on the AFW data set

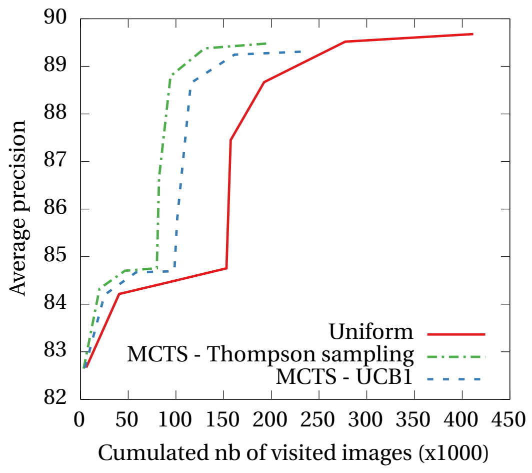


(b) Performance on the PASCAL face data set

Figure 4.9 – Performance of the DPM face detector. Our MCTS strategies (averaged over 4 runs) lead to the same accuracy as the uniform approach. The black curve is the one of Mathias et al. (2014) and is here to show that our implementation reaches state-of-the-art accuracy. As explained in section 2.2.5, the AP score can be misleading because curves can cross one another. The use case of the detector would be in [0.8, 0.85] on figure 4.9b for which our implementation leads to higher performances, although the overall AP score is lower.

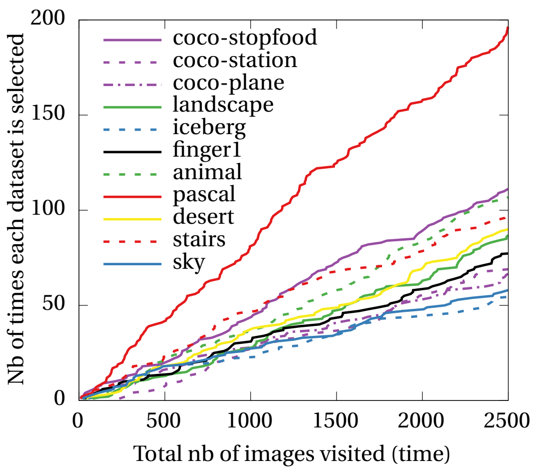


(a) Evolution of the performance on AFW

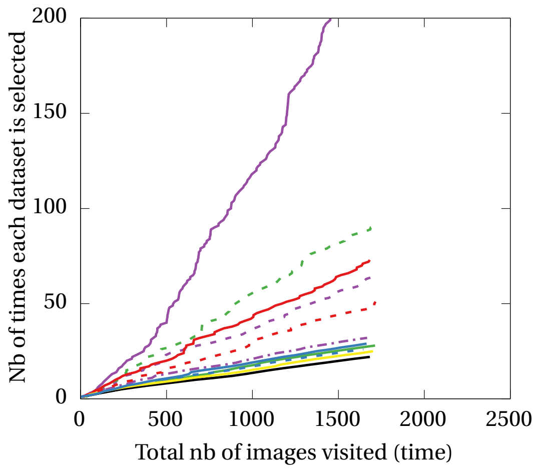


(b) Evolution of the performance on PASCAL

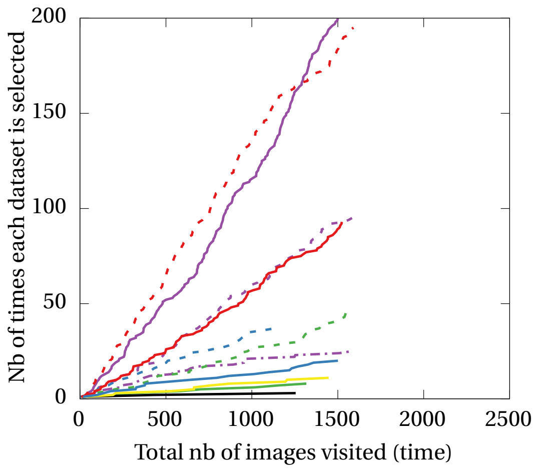
Figure 4.10 – Evolution of the average precision (AP) score as the relabelling steps are performed. The MCTS strategies reaches top performance earlier than the uniform baseline.



(a) Traditional uniform selection



(b) MCTS-UCB1-dense (dense-clust)



(c) MCTS-Thompson (ts-clust)

Figure 4.11 – These plots show how often the data sets are visited by the procedure during the second bootstrapping step to collect 5,000 hard samples with the ACF face detector. The uniform method selects a data set proportionally to the number of images in the set (to be uniform over the entire data set) and requires more than 2,500 images to find enough false positives. Our MCTS bootstrapping approaches identify the most promising data sets (COCO, PASCAL, animals) while leaving “out” the less interesting (sky, fingerprints, etc.). Note that “coco-plane” contains many images of plane/bird on blue sky (*i.e.* less dense in hard samples), so within a (structured) data set, MCTS is also able to discard subbranches and visit it less often.

Behaviour of the sampling

Figure 4.11 shows how often the data sets are visited over time when training an ACF face detector. The uniform approach selects images uniformly in the whole data set, that is proportionally to the number of images in each subsets. The PASCAL data set is more visited because it contains 7,537 images while the other sets have around 3,000. Each data set is

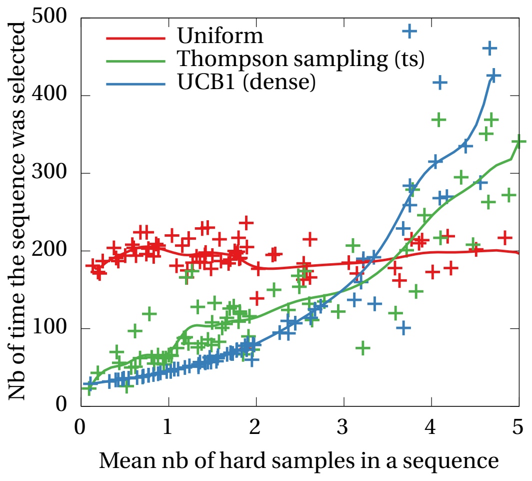
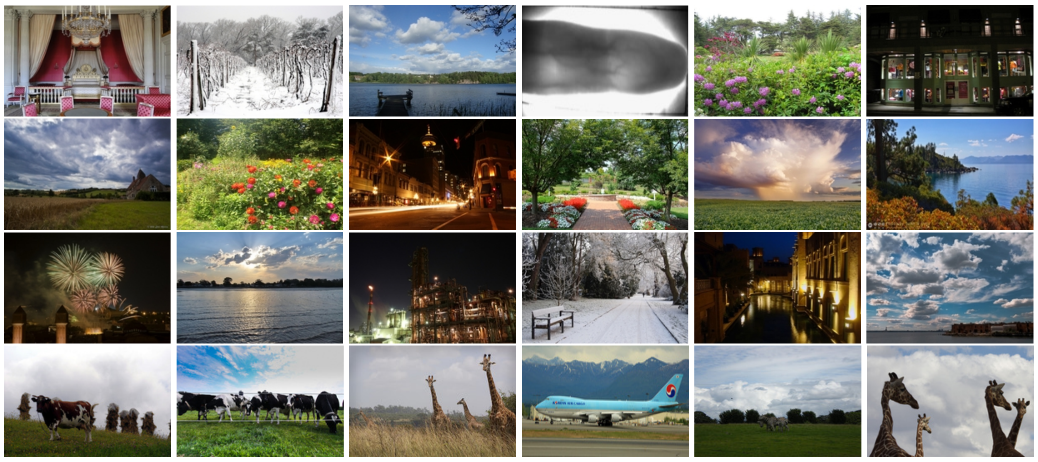


Figure 4.12 – Number of times a sequence is visited by the various strategies. One cross represents one video sequence of the Caltech pedestrian data set, and the lines are an extrapolation for each method. All sequences are equally visited by the uniform approach whereas richer sequences in hard samples are much more selected by the bandit based strategies.

visited linearly with time. Unsurprisingly the MCTS approaches identify data sets PASCAL, COCO, animals or stairs are being rich in hard samples and visit them more often. Data sets of sky, desert of fingerprints are quickly identified as being useless. The speedup is therefore due to the identification of good branches of the data set. Figure 4.13 shows some examples of images which are rarely visited or frequently visited. The images of figure 4.13a were in a branch identified as non promising, as a result of which, they were only selected once. They contain large uniform areas (skies, oceans, etc.), while the images of figure 4.13b were selected 20 times. These images exhibit patterns which are reminiscent of the object of interest (here a face): flowers, animals, circular shapes. MCTS has properly identified the corresponding branches as rich in hard samples.

When MCTS methods are applied on a pre-clustered data set (suffix “grad” and “gist” in table 4.2) the speedup is even more than on non-clustered data set. This is due to the fact that in COCO and PASCAL sets, the pre-clustering puts uninformative images in the same clusters, such as planes over a blue sky or landscapes. MCTS thus identifies uninformative subbranches. We did not use this pre-clustering step on the Caltech pedestrian data set because the temporal structure is consistent in itself. As a sanity check, MCTS on a shuffled data-set (suffix “shuf”) performs as poorly as the uniform approach.

Figure 4.12 shows how many times a sequence of the Caltech pedestrian set was selected as a function of its average number of hard samples. On the graph, one point is a sequence. Sequences with few hard samples (left part of x-axis) are less visited, while sequences with more hard samples are visited more often (right part). MCTS methods have concentrated their sampling on rich sequences hence the speedup.


(a) Images rarely selected

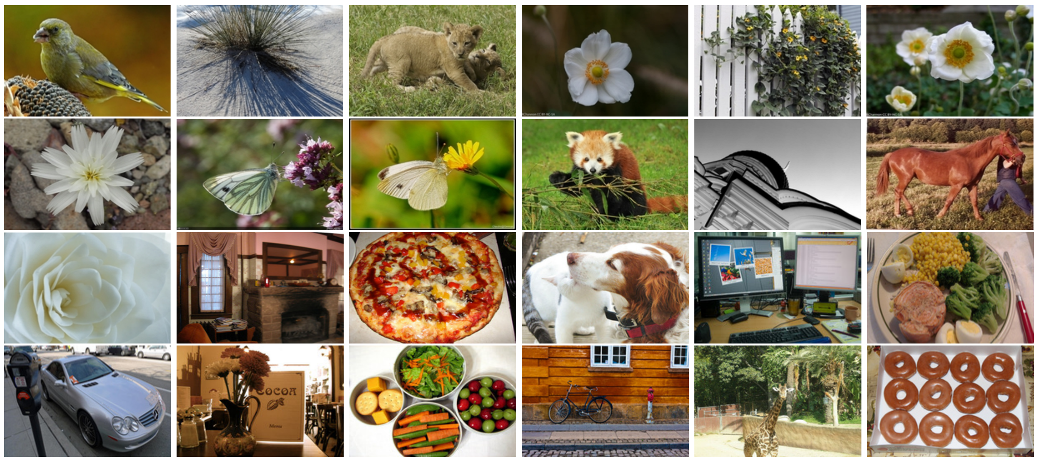

(b) Images frequently selected

Figure 4.13 – Mined negative images to find hard samples to train a DPM face detector. The images of figure 4.13b were revisited 20 times, while images of figure 4.13a were only revisited once.

4.5 Discussion

MCTS bootstrapping presented in this chapter has shown to be an effective strategy to identify promising subgroups of images that are rich in hard samples required to train object detectors.

The image collection should first be organised in a consistent tree structure that serves as a support for the Monte Carlo tree search. The images can either be grouped by key words (sky, animals, landscape, etc.) or clustered directly from their thumbnails for fast computing. Since similar images are equally rich in hard samples, the MCTS procedure is able to quickly identify which part of the tree contains informative images, and which ones do not. MCTS therefore eliminates subbranches of images without looking at all of them.

We can name several limitations of the method here. The first one is that the collection should not contain the object of interest. However, this is the case for any procedure that requires to bootstrap object detectors. This can be done either by using annotated images, or with active learning, where the user would label a few images proposed by the learner. The second one is that MCTS would not bring much speedup on a small collection of images, because the learner would visit them all anyway to get the targeted number of hard samples; MCTS would simply cause them to be visited in a different order. MCTS bootstrapping does not suit sets whose images are homogeneously informative either. In this case, there would not be any uninformative images to discard, and MCTS would be as good as the uniform approach.

5 Importance Sampling Tree for Large-Scale Empirical Expectation

Contents

This chapter proposes a tree-based procedure inspired by the Monte Carlo tree search that dynamically modulates an importance-based sampling to prioritise computation, while getting unbiased estimates of weighted sums. This method is applied to learning on large training sets, and to the evaluation of large-scale SVM. This chapter is based on the following publication:

Olivier Canévet, Cijo Jose, and François Fleuret. Importance sampling tree for large-scale empirical expectation. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2016

5.1 Introduction

Most of machine learning algorithms require computing empirical expectations from the data, either to estimate a loss, like in the case of neural networks or boosting, or to compute the response of a predictor, like for support vector machines. And as usual in machine learning, more data yields more accurate models, which is done by enriching the training sets with artificial distortions (as we have presented in section 2.3).

An empirical expectation takes the form of a sum of a quantity evaluated on many data points, and in practice most of these summed terms are negligible. In training, most of the samples are far from the boundary between classes, and as a result, the classifier is confident about their label. For boosting, it translates into a small boosting weight for these points, while for neural networks, it results in a very small loss. For kernel support vector machines, at test time, many terms of the sum are negligible because the prediction on a test point is only modulated by its immediate neighbours in the training set. For instance, a Gaussian kernel for support vector machine can produce very small kernel values between the test points and the support vectors.

Figure 5.1 shows the cumulated training loss of a convolutional neural network after one epoch on the MNIST data set. Half of the loss is carried by 700 samples out of 60,000, which is 1.2%. To make the chart, we computed the loss on the training samples, and sorted them.

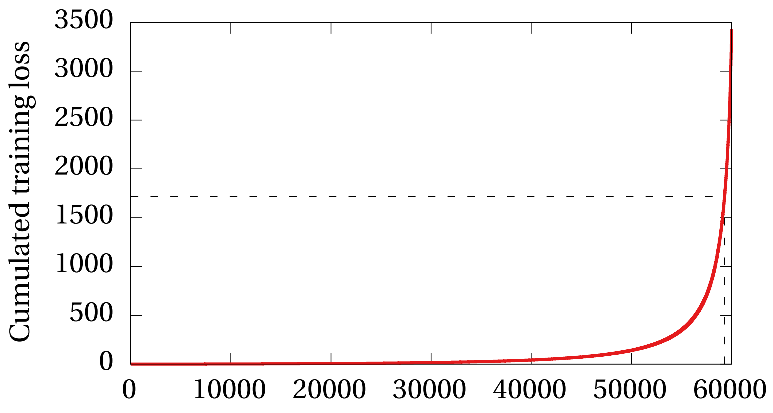


Figure 5.1 – Cumulated training loss of a convolutional neural network after one epoch on the MNIST data set

Despite this observation, algorithms still rely on an exhaustive loop through the samples. Some approaches have been developed to prioritise samples after they have already been seen (Kalal et al., 2008; Fleuret and Geman, 2008), or in subsets sampled uniformly (Loosli et al., 2007), but they do not use structures given *a priori* over the said samples, combining it with statistical observations made over those already observed to reject groups without looking at them.

Given the fact that some training samples are more important than others, we would like to efficiently exploit this observation to either accelerate training, or to improve it. This chapter presents a new structure, called Importance Sampling Tree, which is a binary tree whose branches contains training samples consistent with one another, and which samples leaves (*i.e.* training points) by biasing towards more important points while providing a correcting factor compensating for its sampling bias.

5.2 Related work

5.2.1 Careful selection of training samples

This section describes how it can be important to carefully select training samples, or how their contribution should be modulated in order to improve the training phase.

In the case of boosting, the reservoir boosting introduced by Lefakis and Fleuret (2013) for a memory constrained environment is an example of method which selects training samples. The learner cannot use all the samples to choose the next weak learner but can only use a subset called the reservoir. At each iteration, the reservoir is augmented with new random samples, then reduced down to its original size which is finally used to choose the weak learner. The authors proposed an efficient approach based on the projection of the samples in the classifier space to retain the maximum amount of information of the entire set. In the end, the predictor is learnt with a more representative set of samples of the full training set.

Regarding support vector machines, Bordes et al. (2005) proposed an online algorithm, LASVM (described in more detail in section 6.2.2). By combining it with Active Selection, which consists in training with samples close to the current boundary, they show that the training time can be reduced, and that this results in more compact models with fewer support vectors, while reaching equivalent or superior accuracy compared to standard algorithms. In this case, the selection of training samples close to the boundary is the key element to achieve a better training.

The following works analyse how the training samples can be modulated and re-weighted in the learning process.

The works of Fleuret and Geman (2008) and Kalal et al. (2008) are concerned with subsampling and boosting. Subsampling has usually been used when training predictors on very large data sets, where batch learning is very difficult or even infeasible. These two works show that a smart sampling, as opposed to a uniform sampling, has an impact on the performance of the final classifier. In the boosting procedures proposed by Fleuret and Geman (2008) and Kalal et al. (2008), training samples are selected to train a weak learner based on their boosting weights. The classifier is presented with highly misclassified samples, with a high individual weight, but also with representatives of populations of low weighted individual samples which have a large cumulative weight. This procedure yields a better convergence of the learner.

Recently, importance sampling has been studied in the context of stochastic gradient descent (Zhao and Zhang, 2014; Needell et al., 2013). The traditional uniform selection of the training samples always yields an unbiased estimate of the true gradient, but its variance might be large. These works use importance sampling, which has been known to specifically reduce the variance of the estimated quantity. The two works show that, under certain conditions, importance sampling can improve the convergence rate of stochastic gradient descent algorithms.

Following this line of research, the next section briefly presents what importance sampling is, and how the variance of the estimators is reduced.

5.2.2 Monte Carlo estimation and Importance Sampling

The core idea of Monte Carlo methods is to reformulate the original problem into the estimation of an expectation. The estimation is then performed by random sampling. Monte Carlo methods were first used in the field of statistical physics to approximate the value of intractable integrals. And in the previous chapter, we have used Monte Carlo tree search, in which the exploration of the tree was done through simulation at each step.

This section introduces the basics of importance sampling and how it can improve the convergence of Monte Carlo methods by reducing the variance of the estimators.

Monte Carlo methods are used when it comes to computing

$$\mu = \mathbb{E}_p[f(X)] = \int_{\mathcal{D}} f(x)p(x)dx, \tag{5.1}$$

where $f : \mathcal{D} \to \mathbb{R}$ is a function, $X \sim p$ is a random variable on $\mathcal{D}$, whose probability density function p is zero outside $\mathcal{D}$.

In the basic setting of Monte Carlo, μ is estimated by using the following “Crude Monte Carlo” estimator,

$$\hat{\mu}_{MC} = \frac{1}{N}\sum_{n=1}^{N} f(X_n), \tag{5.2}$$

where $X_1, \ldots, X_N$ are independent and identically distributed random variables from p. The expectation of estimator $\hat{\mu}_{MC}$ is

$$\mathbb{E}[\hat{\mu}_{MC}] = \mathbb{E}_p[f(X)] = \mu, \tag{5.3}$$

and its variance is

$$\mathbb{V}_p[\hat{\mu}_{MC}] = \mathbb{E}_p[\hat{\mu}_{MC}^2] - \mathbb{E}[\hat{\mu}_{MC}]^2 = \int_{\mathcal{D}} f^2(x)p(x)dx - \mu^2. \tag{5.4}$$

The idea of importance sampling is to introduce a new probability density q, with support $\mathcal{Q}$

such that $q(x) \neq 0$ when $p(x) \neq 0$, and rewrite

$$\mu = \mathbb{E}_p[f(X)] = \int_{\mathcal{D}} f(x)p(x)dx = \int_{\mathcal{D}} \frac{f(x)p(x)}{q(x)} q(x)dx = \mathbb{E}_q\left[\frac{f(X)p(X)}{q(X)}\right]. \tag{5.5}$$

The problem is reformulated as the estimation of the expectation of a new random variable when $X \sim q$. The estimation is performed using the following estimator:

$$\hat{\mu}_{IS} = \frac{1}{N}\sum_{n=1}^{N} \frac{f(X_n)p(X_n)}{q(X_n)}. \tag{5.6}$$

By construction, the expectation of $\hat{\mu}_{IS}$ is

$$\mathbb{E}[\hat{\mu}_{IS}] = \mu, \tag{5.7}$$

and its variance is

$$\begin{aligned} \mathbb{V}_q[\hat{\mu}_{IS}] &= \mathbb{E}[\hat{\mu}_{IS}^2] - \mathbb{E}[\hat{\mu}_{IS}]^2 = \int_{\mathcal{Q}} \frac{f^2(x)p^2(x)}{q^2(x)} q(x)dx - \mu^2 \\ &= \int_{\mathcal{D}} \frac{f^2(x)p^2(x)}{q(x)} dx - \mu^2 \quad \text{(same support for simplicity)} \end{aligned} \tag{5.8}$$

which can be rewritten as

$$\mathbb{V}_q[\hat{\mu}_{IS}] = \int_{\mathcal{D}} \frac{f^2(x)p^2(x)}{q^2(x)} q(x)dx - \left[\int_{\mathcal{D}} f(x)p(x)dx\right]^2 \tag{5.9}$$

By using the Cauchy-Schwarz inequality, we can look for a particular q to minimise the variance $\mathbb{V}_q[\hat{\mu}_{IS}]$. The Cauchy-Schwarz inequality states that for two integrable real functions ϕ and ψ,

$$\left[\int \phi(x)\psi(x)dx\right]^2 \leqslant \int \phi^2(x)dx \int \psi^2(x)dx. \tag{5.10}$$

Here, by using

$$\phi(x) = \frac{f(x)p(x)}{\sqrt{q(x)}} \quad \text{and} \quad \psi(x) = \sqrt{q(x)}, \tag{5.11}$$

we have

$$\left[\int \frac{f(x)p(x)}{\sqrt{q(x)}} \sqrt{q(x)}dx\right]^2 \leqslant \int \frac{f^2(x)p^2(x)}{q(x)} dx \underbrace{\int q(x)dx}_{=1}. \tag{5.12}$$

We therefore have

$$\mathbb{V}_q[\hat{\mu}_{IS}] = \int \frac{f^2(x)p^2(x)}{q(x)} dx - \left[\int f(x)p(x)dx\right]^2 \geqslant 0. \tag{5.13}$$

According to the Cauchy-Schwarz inequality, the inequality becomes an equality when the two functions are proportional. Here the variance can be made minimal (actually zero) if q is proportional to fp. Using q proportional to fp might not always be possible, but this shows that by choosing q "similar" to fp, the variance can be reduced, and importance sampling will focus on the part of the domain where fp is large. In practice, q can be chosen so as to get a smaller variance than the "Crude Monte Carlo" estimator: using equations 5.4 and 5.8,

$$\begin{aligned}\mathbb{V}_p[\hat{\mu}_{MC}] - \mathbb{V}_q[\hat{\mu}_{IS}] &= \int_{\mathcal{D}} f^2(x)p(x)dx - \int_{\mathcal{D}} \frac{f^2(x)p^2(x)}{q(x)}dx \\ &= \int_{\mathcal{D}} f^2(x)\left(1-\frac{p(x)}{g(x)}\right)p(x)dx \qquad (5.14)\end{aligned}$$

Using q "similar" to fp as a probability distribution for the observations X_n means that we would like to focus on regions of $\mathcal{D}$ where fp has high values. Importance sampling therefore samples in regions of high values, and at the same time compensates for the induced bias (equation 5.6).

5.2.3 Discrete sampling

In this section, we present various ways to sample according to a discrete distribution. Figure 5.2 aims at illustrating the methods, and figure 5.2a is an example of a non normalised discrete distribution. Given a set of N positive weights $w_1, w_2, \dots, w_N$, we are interested in sampling according to the corresponding distribution, the density of which is

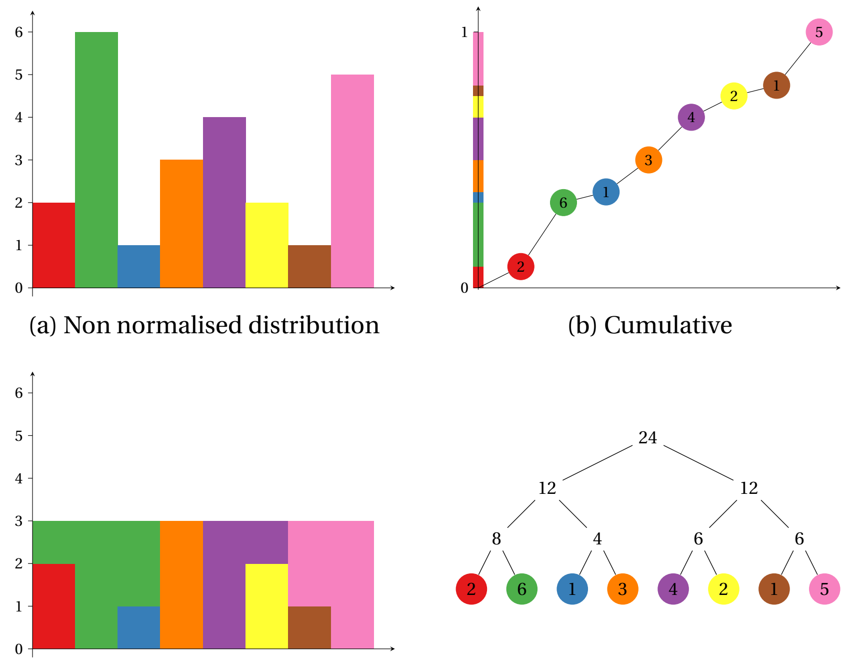
$$f(n) = \tilde{w}_n = \frac{w_n}{\sum_m w_m}. \qquad (5.15)$$

Inverse Transform Method (discrete version)

The first method that is taught at school to generate a random variable from a discrete distribution is the inverse transform method. It consists of a preprocessing step which builds the cumulative distribution (figure 5.2b). Then, the sampling method is divided into two stages: the first one is to generate $u \in [0,1]$ uniformly, and the second is to find which interval of the cumulative it corresponds to, that is finding n such that

$$\sum_{i=1}^{n-1} \tilde{w}_i \leqslant u \leqslant \sum_{i=1}^{n} \tilde{w}_i. \qquad (5.16)$$

Building the cumulative distribution is $\mathcal{O}(n)$ and finding the interval u falls into can be $\mathcal{O}(n)$ in a naive version, or $\mathcal{O}(\log n)$ if the search is done with a tree.



(a) Non normalised distribution (b) Cumulative

(c) Alias (d) Tree Sampling

Figure 5.2 – Various ways to sample according to a discrete distribution

Alias Method

The alias method was introduced by Walker (1974) and is based on the fact that all discrete distributions can be expressed as a mixture of two-point distributions. As a preprocessing step (figure 5.2c), the distribution is rearranged in N buckets of size $1/N$ (W/N when the weights are not normalised as on the figure), each bucket being made of two weights. In practice, to rearrange the distribution, one can use two stacks, one containing weights smaller than $1/N$, and the other containing weights greater than $1/N$. Building one bucket consist in picking one weight smaller than $1/N$ and "filling the rest" of the bucket with a part of a weight larger than $1/N$. This preprocessing time can be done in $\mathcal{O}(N)$.

The sampling procedure is now very simple. First, one selects uniformly a bucket (*i.e.* generate a number in $[\![1, N]\!]$) and second, considering this bucket (made of two weights), one generates a number in $[0, 1/N]$ to sample one of them. The sampling procedure is therefore made in $\mathcal{O}(1)$.

Roulette Wheel Selection

The third sampling method that we describe is known as the roulette wheel selection. The underlying structure is a binary tree which carries at its leaves a weight w_n, and its internal nodes carries the sum of the weights below (see figure 5.2d). Sampling now consists in recursively going down the tree by "choosing left or right" with the corresponding probability:

$w_n \in \mathbb{R}_+$	Positive weights we want to approximate through sampling
N	Number of weights
$\mathbf{N}$	A random variable on $[\![1, N]\!]$
$\mathcal{T}, \mathcal{L} \subset \mathcal{T}$	Nodes and leaves of our sample tree
D	The depth of the tree
$c_0(x), c_1(x)$	Children of node x
U	Minimum number of observations we impose before biasing the θ_x
$\mathcal{L}(x) \subset \mathcal{L}$	Set of leaves of the sub-tree whose root is x
$n(x) \in [\![1, \dots, N]\!]$	Index of the weight at leaf x
$\Theta = (\theta_x)_{x \in \mathcal{T} \setminus \mathcal{L}}$	Probabilities to chose the right sub-tree at each node during sampling
$\mu_\Theta(n; x)$	Probability to reach leaf n given that the sampling passes through node x
$w(x) = \sum_{y \in \mathcal{L}(x)} w_{n(y)}$	Sum of the weights of the leaves in the sub-tree whose root is x
$v(x)$	Number of times the sampling has been through node x
$s(x)$	Sum of the individual estimates of $w(x)$
$\hat{w}_x = s(x)/v(x)$	Estimate of $w(x)$
$\mathcal{S}(x)$	Statistics accumulated at node x

Table 5.1 – Notations

at an internal node whose left (resp. right) child carries w_l (resp. w_r), one selects the left one with probability $w_l/(w_l + w_r)$ and the right one with probability $w_r/(w_l + w_r)$. The tree can be constructed in $\mathcal{O}(N)$ and the sampling is in $\mathcal{O}(\log N)$.

The roulette wheel selection has an advantage over the two previous methods in the case of changing distributions, that is when one or more w_n change. In such a case, if one weight changes, the tree can be updated by setting the new value at the corresponding leaf and by updating all the intermediate internal nodes between that leaf and the root, which result in a $\mathcal{O}(\log N)$ complexity. Whereas in the two previous cases, updating the distribution would require to recompute the cumulative distribution for the inverse transform method, and to recompute the buckets for the alias method, which in both case is $\mathcal{O}(N)$.

The underlying structure of IST is this binary tree, which will be used to select the training samples according to their weights (*i.e.* their importance), but also to get an unbiased estimate of the total weight below an internal node.

5.3 Importance Sampling Tree

This section presents our Importance Sampling Tree (IST). As we have shown in section 5.2.1, focusing on important samples (defined as having a high gradient norm for instance), allows to improve the learning of a predictor. IST is in the same line of research and is designed to focus on high weight samples. Table 5.1 summarises the notations used in the sequel.

5.3.1 Weighted sums for prediction

Our motivation comes from the observation that many important data-driven quantities in machine learning can be expressed by a weighted sum

$$\sum_{n=1}^{N} w_n f(n), \tag{5.17}$$

where $w_n \in \mathbb{R}_+,\ n = 1, \ldots, N$ are positive weights, and $f : [\![1, N]\!] \to \mathbb{R}^D$ a function. And in many cases, N is too large to allow an exhaustive visit of all the weights.

As an example of such quantities, we can name:

- The edge of a weak-learner in AdaBoost

$$\sum_n \underbrace{\exp(-y_n \psi(x_n))}_{w_n} \underbrace{y_n h(x_n)}_{f(n)} \tag{5.18}$$

- The gradient for training a neural network

$$\sum_n \nabla_\alpha l(\psi(x_n; \alpha), y_n) = \sum_n \underbrace{\|\nabla_\alpha l(\psi(x_n; \alpha), y_n)\|}_{w_n} \underbrace{\frac{\nabla_\alpha l(\psi(x_n; \alpha), y_n)}{\|\nabla_\alpha l(\psi(x_n; \alpha), y_n)\|}}_{f(n)} \tag{5.19}$$

- The evaluation of a SVM in its dual form

$$\sum_n \alpha_n y_n K(x_n, x) = \sum_n \underbrace{\alpha_n K(x_n, x)}_{w_n} \underbrace{y_n}_{f(n)}. \tag{5.20}$$

In these examples, a weight can be interpreted as the "importance" of a sample, and in many practical situations, the vast majority of them are negligible (see figure 5.1). We aim at devising an approach – relying on a prior tree structure on the weights – that first, balances computation proportionally to the weights themselves, and second, does so by looking at a fraction of the full family of weights, opening the way to extremely large samples sets.

5.3.2 Monte Carlo estimation of a weighted sum

Given an arbitrary distribution μ on $[\![1, N]\!]$ which puts non-zero probabilities on all the values, and as we have shown earlier, equation (5.17) can be rewritten as

$$\sum_n \mu(n) \frac{w_n}{\mu(n)} f(n) = \mathbb{E}_{\mathbf{N} \sim \mu} \left[\frac{w_{\mathbf{N}}}{\mu(\mathbf{N})} f(\mathbf{N}) \right] \tag{5.21}$$

which we can approximate by generating $\mathbf{N}_1, \ldots, \mathbf{N}_K$ independent, identically distributed following μ, and using the following estimator

$$\hat{m} = \hat{\mathbb{E}}_{n \sim \mu}\left[\frac{w_n}{\mu(n)} f(n)\right] = \frac{1}{K}\sum_{k=1}^{K} \frac{w_{\mathbf{N}_k}}{\mu(\mathbf{N}_k)} f(\mathbf{N}_k). \tag{5.22}$$

Similarly to what we derived in section 5.2.2, we can show that the expectation of the estimator is

$$\mathbb{E}[\hat{m}] = \sum_{n=1}^{N} w_n f(n), \tag{5.23}$$

its variance is

$$\mathbb{V}[\hat{m}] = \frac{1}{K}\left(\sum_{n=1}^{N} \frac{w_n^2 f^2(n)}{\mu(n)} - \left(\sum_{n=1}^{N} w_n f(n)\right)^2\right) \tag{5.24}$$

and that it can be made minimal when

$$\mu(n) = \frac{w_n |f(n)|}{\sum_{k=1}^{N} w_k |f(k)|} \tag{5.25}$$

In our case, the most natural choice for μ to estimate the quantities (5.18), (5.19), and (5.20), would be to use

$$\mu(n) = \frac{w_n}{\sum_k w_k}, \quad (|f(n)| = 1 \text{ in all cases here}) \tag{5.26}$$

in which case we would have our desired property of investing the computation proportionally to the weights, and minimize optimally the variance of our estimator.

This choice makes sense if computing $\sum_k w_k$ is tractable, or so cheap to compute compared to the computation of the $f(n)$s that it still provides a substantial gain. However, we are interested in situations where not only w_n is more expensive to compute than $f(n)$, but we also aim at scaling N up to values far greater than the number of CPU operations we have at our disposal.

5.3.3 Importance Sampling Tree

We propose a novel structure called Importance Sampling Tree (IST), which is a binary tree carrying the weights $w_1, \ldots, w_N$ at its leaves, and having at each internal node statistics about the weights in the leaves below it. Given such a tree, we use a recursive sampling procedure that results in a sampling of the leaves (in the spirit of the Roulette Wheel Selection of section 5.2.3), and – in a manner similar to the MCTS – we update the estimates at the nodes every time a sampling is done, and modulate the sampling policy accordingly.

Let $\mathcal{T}$ be the set of tree nodes, $x^* \in \mathcal{T}$ the root node, $\mathcal{L} \subset \mathcal{T}$ the leaves. Since the leaves carry the weights $w_1, \dots, w_N$, for any leaf $x \in \mathcal{L}$, let $n(x) \in [\![1, N]\!]$ be the index of the weight there. Note that we will often make a confusion between $x \in \mathcal{L}$ and $n(x)$, identifying a leaf with its index.

For any internal node $x \in \mathcal{T} \setminus \mathcal{L}$, let $c_0(x), c_1(x) \in \mathcal{T}$ be its two child nodes. Finally let $\mathcal{L}(x)$ be the leaves of the sub-tree starting at x and $w(x)$ the sum of their weights. Hence, in particular

$$\forall\, x \in \mathcal{L}, \quad w(x) = w_{n(x)} \qquad \text{and} \qquad \mathcal{L}(x) = \{x\}.$$

Figure 5.3 sketches the structure of the tree below node x with the notations introduced above.

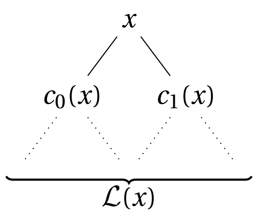


Figure 5.3 – Definition of the children and the leaves of node x

Given a family of "bifurcation probabilities"

$$\Theta = (\theta_x)_{x \in \mathcal{T} \setminus \mathcal{L}} \in]0,1[^{\|\mathcal{T} \setminus \mathcal{L}\|}, \tag{5.27}$$

that is for each node the probability to "go down on the right", we can derive for each node x and each leaf n a probability $\mu_\Theta(n; x)$ to reach the leaf n if we start from x and follow the θs at each node we meet. This probability is the product of the probabilities of the bifurcations to go from x to n. Given a leaf n, the $\mu_\Theta(n; x)$ for all the parents x of n can be computed in $O(D)$, where D is the depth of the tree.

Adaptive sampling

We could use many different policies to modulate the θ_x and bias the sampling according to what we have observed. We propose two strategies, both based on accumulating statistics $\mathcal{S}(x)$ at every node about the weights observed during the previous sampling:

Using empirical weights — Here, the statistics we keep are $\mathcal{S}(x) = (s(x), v(x))$ where $s(x)$ is the sum of the weight estimates $w_n / \mu_\Theta(n; x)$, and $v(x)$ is the number of times the sampling went through x. We make a small notation abuse, because w_n is the weight observed at the time the tree was traversed, and $\mu_\Theta(n; x)$ is the probability at this time. These quantities are constantly changing, and are also different between all nodes, including the leaves. For $v(x) > 0$,

$$\hat{w}(x) = \frac{s(x)}{v(x)} \tag{5.28}$$

is an unbiased estimator of $w(x)$. We set θ_x to the ratio of the number of leaves in the child sub-trees if we do not have enough sampling for proper estimations, and to the ratio of the estimations of the weights otherwise. Formally with U a meta-parameter setting the minimum number of observations we request for biasing:

$$\theta_x = \begin{cases} \dfrac{\|\mathcal{L}(c_1(x))\|}{\|\mathcal{L}(c_0(x))\| + \|\mathcal{L}(c_1(x))\|} & \text{if } \nu(c_0(x)) < U \\ & \text{or } \nu(c_1(x)) < U, \\ \dfrac{\hat{w}(c_1(x))}{\hat{w}(c_0(x)) + \hat{w}(c_1(x))} & \text{otherwise.} \end{cases} \tag{5.29}$$

Using weight intervals — In this case, $\mathcal{S}(x) = (l(x), u(x))$, corresponding to lower and upper bounds on $w(x)$, which are updated and getting tighter after every sampling.

We define $\Phi(x)$, a function of the upper bound $u(x)$ and lower bound $l(x)$ at node x, to reflect the importance of a node, and we set θ_x to be

$$\theta_x = \frac{\Phi(c_1(x))}{\Phi(c_0(x)) + \Phi(c_1(x))} \tag{5.30}$$

This second strategy can be interesting when we have some information *a priori* on the weights. This is the case in particular for support vector machines, because the kernel values can be bounded. In the experiments of section 5.4.3, we compare several definitions of Φ.

Hence, if we need T samples, for $t = [\![1, T]\!]$:

1. Recursively sample a path down the tree to a leaf n^t, according to $\mu_{\Theta^t}(\,.\,; x^*)$.

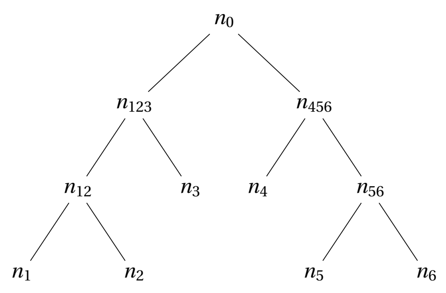
2. For every node x visited on that path compute $\mathcal{S}^{t+1}(x)$ and θ_x^{t+1} according to equation (5.29) or (5.30) depending on the application. All other nodes remain unchanged.

Then, following section 5.3.2, we can use

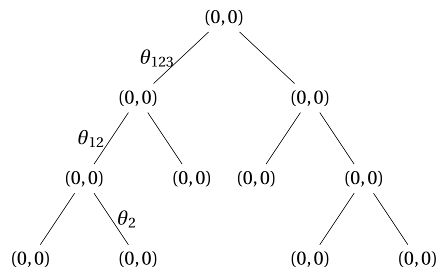
$$\sum_{n=1}^{N} w_n f(n) \simeq \frac{1}{T} \sum_{t=1}^{T} \frac{w_{n^t}}{\mu_{\Theta^t}(n^t; x^*)} f(n^t). \tag{5.31}$$

Note that the Θ^ts in that expression – and the resulting μs – are the ones that were in the tree when each sampling was done. Also, note that the update of $\mathcal{S}$ and Θ can be delayed, suppressed, or done in an arbitrary order if implementation constraints impose it.

The following figures illustrate in a very simple case the various computations and updates of the statistics accumulated at the nodes. We consider the case when IST uses empirical weights as importance weights (equation 5.29).

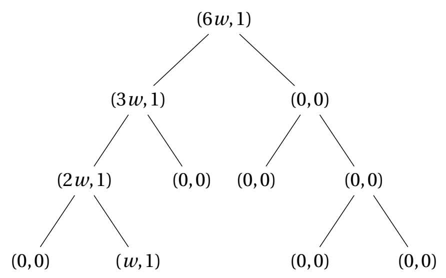


In this example, the tree has 6 leaves (from 1 to 6). We name the internal nodes by concatenating the indices of the leaves below (*i.e.* the parent of n_5 and n_6 is n_{56}), and we call n_0 the root. As described before, each node x maintains statistics $(s(x), v(x))$, where $s(x)$ is the sum the weight estimates below, and $v(x)$ is the number of times the sampling went through x.



Initially, all statistics are set to zero, and as not enough observations have been made yet ($v(x) < U$ for all nodes, equation 5.29), the sampling is uniform over the number of leaves. Here $\theta_{123} = 3/6$ because there are 3 leaves below n_{123} and 6 leaves below the root. Similarly, $\theta_{12} = 2/3$, and $\theta_2 = 1/2$.

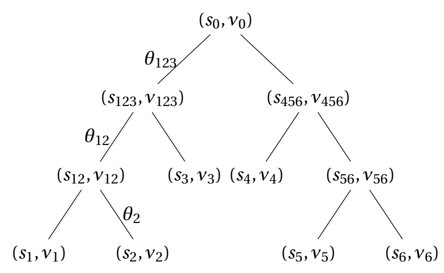
Leaf n_2 was sampled, that is we select training sample n_2 to train the classifier. We suppose that the weight of sample n_2 is w (a boosting weight, a gradient norm, etc.). The training sample is modulated by $w/\theta_{123}\theta_{12}\theta_2$ in the sum of equation 5.31. And finally, the IST is updated by recursively backpropagating the observation. We have:



- $s_{12} = \frac{w}{\theta_2} = 2w$
- $s_{123} = \frac{w}{\theta_2}\frac{1}{\theta_{12}} = 3w$
- $s_0 = \frac{w}{\theta_2\theta_{12}}\frac{1}{\theta_{123}} = 6w$

We also increment the number of visits, namely v_2, v_{12}, v_{123}, and v_0. At the root, an unbiased estimate of the total weight below is $6w/1 = 6w$, which is as if all the leaves carried the same weight. This is the best estimation given the single observation made so far.

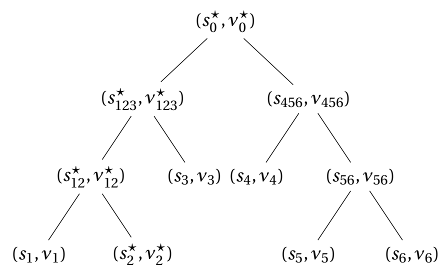
We now suppose that the sampling reached a point where all nodes have been visited at least U times, that is $v(x) > U$ for all nodes. Note that at a given iteration, the sampling policy can be biased in the top levels of the tree (where nodes are such that $v(x) > U$), and uniform in the bottom when there are not enough observations. Here, for the sake of simplicity, we assume that all nodes have at least U visits, which in particular implies that the tree was sampled at least $6U$ times.



Starting from the root, the IST policy uses the weight estimates to bias the sampling. Here, we have

- $\theta_{123} = \frac{s_{123}/v_{123}}{s_{123}/v_{123}+s_{456}/v_{456}}$
- $\theta_{12} = \frac{s_{12}/v_{12}}{s_{12}/v_{12}+s_3/v_3}$
- $\theta_2 = \frac{s_2/v_2}{s_1/v_1+s_2/v_2}$

We recall that s_n/v_n is an unbiased estimate of the total weight below node n. At a node, the probability to "go right" is the ratio between the right weight estimate and the sum of the weight estimates of both children.



Node n_2 was sampled and we call the observed weight w_t. We denote with a $\star$ the statistics that are changed at this iteration, namely

- $s_2^\star = s_2 + w_t$
- $s_{12}^\star = s_{12} + \frac{w_t}{\theta_2}$
- $s_{123}^\star = s_{123} + \frac{w_t}{\theta_2}\frac{1}{\theta_{12}}$
- $s_0^\star = s_0 + \frac{w_t}{\theta_2\theta_{12}}\frac{1}{\theta_{123}}$

And as before, the number of visits are simply incremented ($v_2^\star = v_2 + 1$, etc.).

Building the tree

When using the roulette wheel selection (where the weights are known and fixed), the leaves need not be organised in a particular way. One leaf with a large weight and one with a small weight can share the same parent in the tree. The accumulated statistics in the tree accounts for this discrepancy, and will in the end, properly allow to sample from the distribution made of the weights.

In our case, the weights are zero initially because no observation has been made, and as we sample the tree and train the classifier, the weights of the leaves are progressively observed.

With a view to reducing the variance of our estimations even more, we organise the leaves in a particular way. In practice, we observe that similar samples behave the same way in regard of the classifier, when the latter is not to noisy: the neighbour of a high loss sample will also have a high loss, the neighbour of a highly negative sample will also be negative.

Therefore, we want to organise the tree by putting similar samples in the same branches. In the end, we want the tree to reflect the statistical regularities of the training samples through the classifier. When such a tree is sampled and updated with weights, the accumulated statistics at an internal node should have a lower variance, and the sampling should be biased towards “good” branches (with high weights).

In our experiments, all the data sets are related to the Euclidean metric. IST is therefore simply built by recursively applying a top-down 2-means clustering, until reaching a cluster of two samples: we first cluster the whole training set in two clusters, which gives the two children of the root; then each of these two clusters are themselves clustered in two, etc. The deeper the clustering, the more similar the samples.

5.4 Experiments

5.4.1 Adaboost on a 2D synthetic problem

To get a first idea of the behavior of the IST in practice, we use AdaBoost on a simple synthetic problem. This algorithm is a very good candidate since the exponential loss is known to induce strongly unbalanced sample weights. Our objective is to assess if the IST is able to focus the sampling efficiently enough to cope with the divergence of the loss that is classically observed in validation.

The synthetic task is a classification problem where the signal X is uniform in the unit square $[0,1]^2$ and the class Y is $+1$ in a disc centered in that square and -1 elsewhere. The total sample set we consider contains 8192^2 points located on a regular grid in the unit square. The binary tree structure we use for the IST (as described in section 5.3.3) recursively splits the x and the y axes, and has a depth of 27, corresponding to 13 splits in each directions and one level for the leaves (this can be viewed as a deterministic 2-means clustering that we described in section 5.3.3).

We consider a standard AdaBoost based on the exponential loss as in equation (5.18), and linear stumps. Each stump is trained by sampling uniformly 100 directions in the 2D plan, and optimising exactly its bias, and its weight in the strong predictor.

We test in our experiments three algorithms:

Boost is the baseline. It samples 1,000 samples uniformly initially, and uses them as a standard training set for all the stumps.

Boost-U re-samples uniformly 1,000 new training samples for each stump.

Boost-IST samples 1,000 training samples with the IST for each stump, and updates the IST with their AdaBoost weights. The same tree is used for all the stumps, and not reinitialised for each of them.

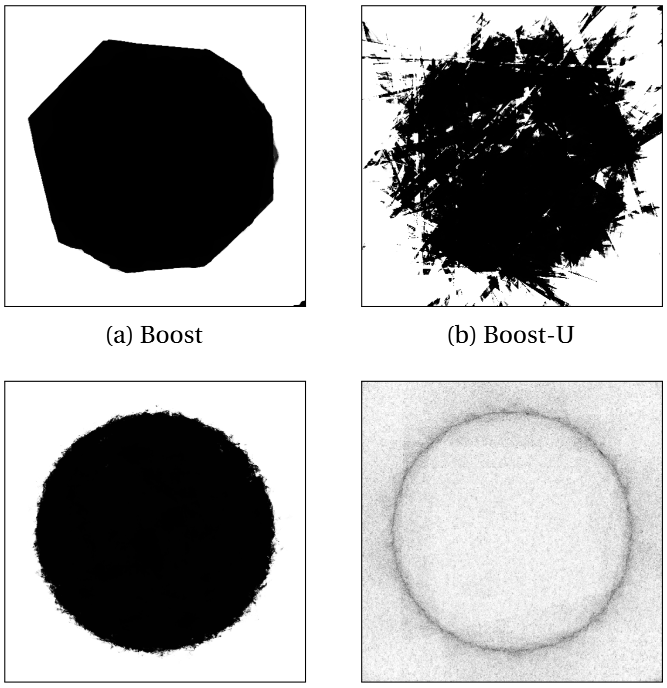


Figure 5.4 – Adaboost on a synthetic 2D problem. Images (a), (b), and (c) depict respectively the prediction obtained with Boost, Boost-U and Boost-IST. Picture (d) shows the sampling intensity with Boost-IST, that is the frequency of position sampled by IST. IST has properly concentrated on the area of interest.

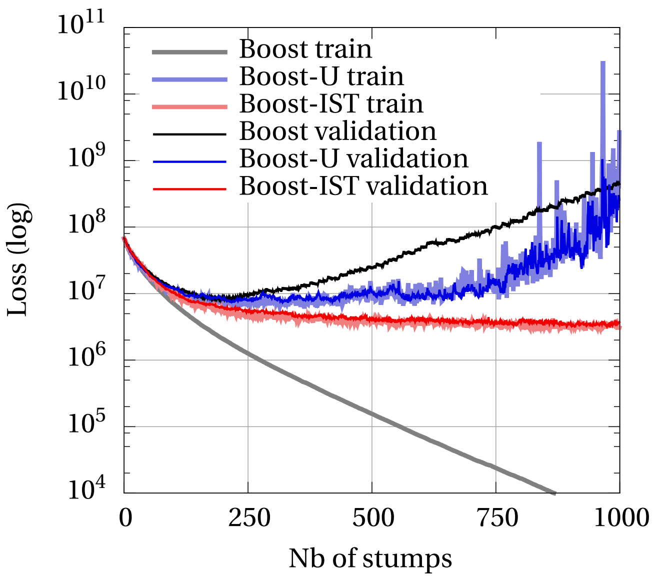


Figure 5.5 – Training and validation losses of AdaBoost on a synthetic 2D problem. The exponential loss behaves pathologically on samples at the frontier which have not been seen during the stump optimization, which impacts both Boost and Boost-U variants, and leads to a diverging training for the latter.

For all variants, we also sample 1,000 samples uniformly initially as a validation set. The results are illustrated in figure 5.4. Here, losses correspond to estimates on the total sample set of 2^{26} samples, hence the initial value of $\simeq$ 6.7e7.

The Boost baseline converges in training and leads to a very sharp and clean prediction decision, albeit only roughly approximating the proper domain, which leads to a diverging validation loss, as the validation samples near the boundary may be misclassified by the predictor, with an "increasingly wrong" prediction when the learning progresses. The Boost-U re-sampling strongly suffers from the same problem, as each re-sampled set contains points strongly misclassified by the current strong learner, which induce very strong sample weights and correspondingly a choice of stumps that gets more and more inconsistent while the process goes on. This results in diverging training and validation losses, and a pathological final strong predictor. The Boost-IST based approach focuses on the boundary population, and by sampling more and more intensively there, it does not diverge and leads to a more accurate decision rule. Remarkably, the validation loss is very well reflected by the training one, even if the former is estimated on a fixed sample set and the latter on one re-sampled at every iteration.

5.4.2 Neural Networks

We assess experimentally in this section how IST can be used to improve the gradient descent procedure to train an artificial neural network.

Multi-layer Neural Network on a 2D synthetic problem

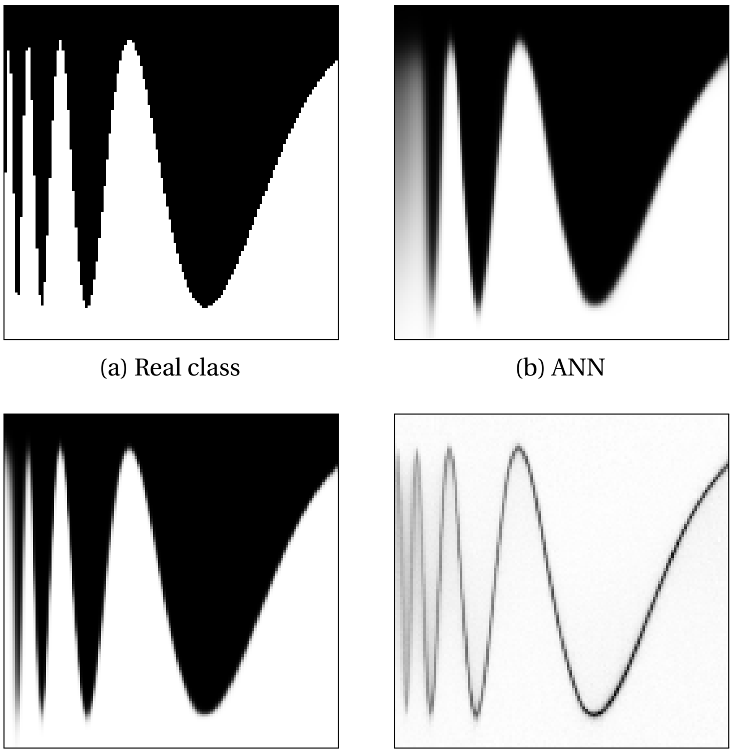
As for the boosting example of the previous section, we consider a 2D synthetic problem, depicted as figure 5.6(a), where the frontier between the two classes is an oscillation of variable frequency. This problem exhibits the difficulty of many real-world data sets in which the core issue is to capture fine details of the boundary.

We train a neural network with two units as input standing for the coordinate in the $[0,1]^2$ domain, two fully connected hidden layers with 40 units each, and one output unit. The transfer function is the hyperbolic tangent, and the weights are initialized layer after layer so that the response of every unit before non-linearity is centred, of standard deviation 0.5. We use the quadratic loss for training, and a pure stochastic gradient descent, one sample at a time. Every 1,000 gradient steps, we compute a validation loss and adapt the step size.

As before, the IST structure recursively splits the $[0,1]^2$ domain in x and y. We compare three strategies:

ANN is the baseline, using uniform sampling in the plane,

ANN-IST samples with IST using the gradient norm as importance function, following equa-



(a) Real class (b) ANN

(c) ANN-IST (d) IST sampling intensity

Figure 5.6 – A multi-layer neural network on a synthetic 2D problem. Image (a) is the binary labelling to learn, (b) is the prediction of the baseline ANN, (c) is the prediction of ANN-IST, and (d) is the sampling density during training with ANN-IST.

tion (5.19) and the same tree structure we use for boosting in section 5.4.1.

ANN-IST-L is the same but uses the loss per sample instead of the gradient norm as the importance function.

We benchmark the three methods through ten train/test runs, with 3 millions gradient steps, and obtain a test error of 2.64%(±0.29%) for ANN, 0.62%(±0.14%) for ANN-IST, and 1.58%(±0.60%) for ANN-IST-L. The losses are consistent with the error rates in all the runs. As shown on figure 5.6, the greater performance of ANN-IST is explained by its ability to capture the thin structure on the left. This behavior is extremely consistent through the runs, and ANN-IST always ranks first, ANN-IST-L second, and ANN third.

Deep Convolution Network

In the second set of experiments, we use a convolutional neural network, which consists of several spatial convolutional layers alternating with pooling layers, and fully connected layers at the end. For the implementation, we use Torch[1] (Collobert et al., 2011), which is a scientific library for luajit containing many implementations of machine learning algorithms, especially convolutional neural networks (CNN).

CNN are trained in an online fashion: at each iteration, a mini batch of a few samples is made (usually 32, 64, or 128 samples), and stochastic gradient descent is applied to update the model parameters. The gradient is computed with equation 5.19.

In Torch, it is not easy to access the gradient of individual samples at each layers, as we previously did in the synthetic example. Therefore, we use the loss as a proxy for the gradient norm: the IST leaves carry the losses observed as the training goes, instead of the gradient norms.

As a sanity check, we first analyse whether selecting the training samples according to their loss is a more effective strategy than the uniform selection. We perform this experiment on the original MNIST data set and compare three strategies:

Uniform is the baseline. The mini batch is populated with samples uniformly taken without replacement from the training set.

True loss is the sanity check using the ideal weight of section 5.3.2. Every 6,000 mini batch, the individual losses of the training samples are computed, and the tree leaves are updated. Leaf weights are not updated in between. Samples are thus sampled with the following probability:

$$\frac{l(\psi(x_n;\alpha), y_n)}{\sum_m l(\psi(x_m;\alpha), y_m)}. \tag{5.32}$$

For the IST experiment (last strategy), we have observed that applying the vanilla IST as described in section 5.3.3 can sometimes be unstable and degenerate. This might be due to bad early estimations, causing IST to over sample a small non representative population of samples. To annihilate this side effect, we introduce a probability ϵ with which a sample is selected uniformly from the training set. $\epsilon = 0$ is the vanilla IST (which sometimes degenerates). The last strategy is therefore:

IST populates the mini batch by selecting a sample uniformly with probability ϵ, or according to IST with probability $1-\epsilon$. The tree weights of the selected samples are updated after each gradient step. We use $\epsilon = 0.5$. So on average, half of the samples of a mini batch are selected uniformly, and the other half with IST.

The three strategies are benchmarked over 8 runs and we analyse the evolution of the training loss (figure 5.7a) and the test accuracy (figure 5.7b). For the baseline, the training loss decreases

[1]`https://github.com/torch/`

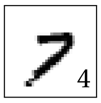
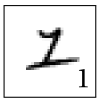
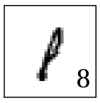
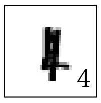
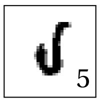
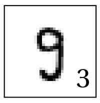
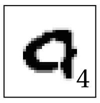
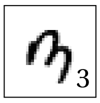
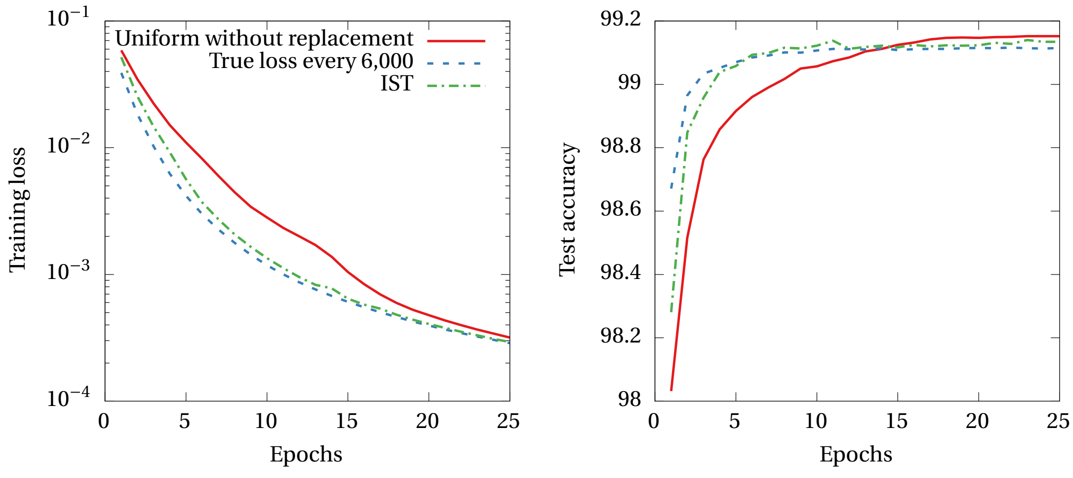
								
Nb sampled	3719	1479	1268	1264	1220	1132	1064	1030
	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)

Table 5.2 – Top eight images that have been sampled many times over 25 epochs by the IST method. The true label of the digits is indicated at the bottom right of the images, that is the label used during training. The number of times the digits were sampled by IST. The uniform without replacement would have selected them 25 times only (1 time per epoch). IST has focused on problematic samples such as wrongly annotated digits (digit (a) and (f)), rotated digits (digit (h)), and gribbles (digit (c) and (e)).



(a) Training loss (b) Test accuracy

Figure 5.7 – A CNN on the MNIST data set. The training loss is reduced much faster by sampling the points according to their loss. Computing the true loss every 6,000 batch is costly but serves as a sanity check to show that the loss is a good proxy for the importance of a point.

as more epochs are performed, and the test accuracy increases and reaches 99.01% after 8 epochs. When the samples are selected according to their loss weight and their gradient rescaled by their probability of being sampled, the training loss decreases much faster, and the test accuracy reaches 99.03% after only 3 epochs. This experiment is of course very costly because it requires to compute the true loss of individual samples every 6,000 mini batches, but demonstrates that selecting the samples proportionally to their loss is an efficient way of finding informative samples.

Table 5.2 shows the samples which are more selected by IST. The uniform baseline would have selected these samples 1 time per epoch (25 epochs in total), but IST has selected them more than 1,000 times. We see that IST concentrates on problematic samples, such as labelling errors, rotated samples, and gribbles.

We now evaluate the IST method on a larger data set. We use the InfiMNIST set presented

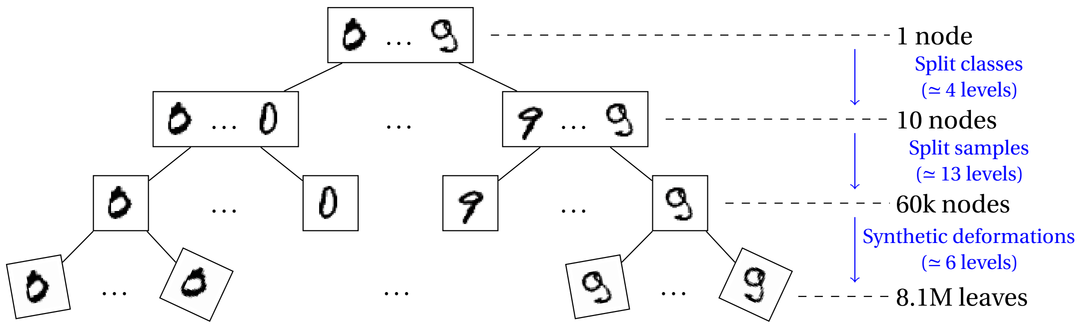
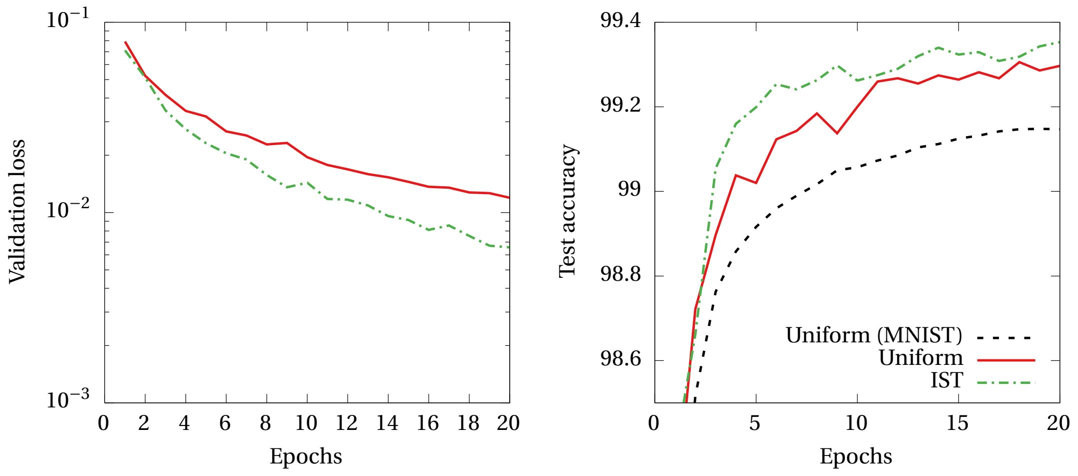


Figure 5.8 – Image tree of distortions. The original images are first split by classes at the top, then recursively clustered using a 2-means algorithm, and finally, the tree of original images is extended at the bottom, with the corresponding distortions.



(a) Validation loss

(b) Test accuracy

Figure 5.9 – A CNN on the InfiMNIST data set. IST decreases the validation loss faster and yields a better test accuracy. The curve on the original MNIST has been reproduced for comparison purposes: the black dashed line is the same as the red one of figure 5.7b.

in section 2.3 with 134 distortions per original digit, yielding 8.1 million samples. To avoid performing the clustering on 8.1 million samples, we only cluster the original samples, by first grouping the samples by class and then by applying the recursive 2-means algorithm as described in section 5.3.3. Once the tree is built on the original samples, it is extended at the bottom by adding the distortions of the corresponding images (see figure 5.8).

To evaluate the training process, we use a "validation" set consisting of the original MNIST images as well as their first 4 distortions. We compute the loss of the classifier on these samples at regular intervals, as well as the test accuracy. Figure 5.9 shows that IST reduces the validation loss faster than the uniform approach (figure 5.9a) yielding a better test accuracy faster (figure 5.9b).

5.4.3 Non-linear SVM Prediction

The prediction cost of non-linear SVM grows linearly with the number of support vectors which in turn grows linearly with the number of training points (Steinwart and Christmann, 2008). Thus prediction using non-linear SVM trained on large data set is prohibitively expensive. As noted earlier, for non-linear SVM such as with a Gaussian kernel, the support vectors which are far way from the test point contribute very little to the decision function. The goal of this section is to exploit this fact to do very fast prediction of Gaussian SVM with IST.

Building the tree

The tree structure used in this case is done as described in section 5.3.3: we perform a top-down 2-means clustering of the support vectors based on the Euclidean distance. The resulting tree is binary; its leaves are the individual support vectors. In addition to the default clustering, internal nodes carry the centroid corresponding to the clusters made of all the support vectors below, and the radius of the cluster (that is the largest distance between the centroid and the points in that cluster, which is 0 at a leaf).

The simple k-means procedure is relevant in this context because the Euclidean distance explicitly appears in the contribution of a support vector x_n in the decision score of a test sample x, that is

$$\alpha_n y_n e^{-\gamma||x-x_n||^2}, \tag{5.33}$$

where γ is the kernel bandwidth set during training. Our objective is to estimate the predictor response on a test sample without computing the full inner products between the test samples and the support vectors.

The main idea here is to use a weighted sampling based on the bounds we have on the total weights of the support vectors below each node. Using the triangle inequality, we can derive a bound on the weights below that node: given a test sample z, the centroid ω of a cluster of radius ρ, we have

$$\underbrace{e^{-\gamma\left(||z-\omega||+\rho\right)^2}}_{\beta_l} \leq K(z_n, z) \leq \underbrace{e^{-\gamma\left|||z-\omega||-\rho\right|_+^2}}_{\beta_u}, \tag{5.34}$$

and since we consider the α_n positive (the sign is carried by $f(n) = y_n$ equation 5.20)

$$\underbrace{\beta_l \sum_n \alpha_n}_{\substack{\text{Lower bound } l(x) \text{ due to} \\ \text{cluster of centroid } \omega}} \leq \sum_n \alpha_n K(z_n, z) \leq \underbrace{\beta_u \sum_n \alpha_n}_{\text{Upper bound } u(x)} \tag{5.35}$$

The tree is traversed using policy of equation (5.30). At each node, the bounds of both children are queried and the sampling goes on until reaching a leaf. Querying a bound requires the computation of one inner product with the corresponding centroid (*i.e.* $||z-\omega||$), but only once as the computation can be reused later. At a leaf, we have the true value $\alpha_n K(z_n, z)$ of the weight. The bounds from the leaf up to the root can now be updated according to the true weight: the upper (resp. lower) bound $u(x)$ (resp. $l(x)$) will now be $u(c_0(x)) + u(c_1(x))$ (resp. $l(c_0(x)) + l(c_1(x))$). The bounds are thus tighter and in future iteration, the sampling will be based on better bounds. Asymptotically, the sampling is performed on exact bounds, that is on the sum of the weights below. In the case of Gaussian kernel, some upper bounds are drastically reduced after a few samplings which cause many parts of the branches not to be sampled again.

While this policy requires to compute inner products with the centroids as well as with the true support vectors in the leaves, the products can be cached once computed and re-accessed when required, without further computation. The true number of inner products for N support vectors of dimension M is $\mathcal{O}(NM)$ whereas for IST, when sampling T times, the number of inner products is at most $\mathcal{O}(TM \log_2 N)$. $\log_2 N$ is the depth of the tree (the number of centroids visited when performing one sampling). This is an upper bound since many computation can be reused.

Defining the probability of bifurcation

We investigate three strategies to define θ_x (*i.e.* the probability to "go down on the right", or to select child $c_1(x)$). To simplify notations, we call u_0 (resp. l_0) the upper (resp. lower) bound of child 0 of node x (*i.e.* $u_0 = u(c_0(x))$).

Mean bound (`mean`) We define the score of a node to be the mean of the upper and lower bounds.

$$\Phi(x) = \frac{u(x) + l(x)}{2} \text{ so } \theta_x = \frac{l_1 + u_1}{l_0 + u_0 + l_1 + u_1} \tag{5.36}$$

Max bound (`max`) We simply use the upper bound to bias the sampling.

$$\Phi(x) = u(x) \text{ so } \theta_x = \frac{u_1}{u_0 + u_1} \tag{5.37}$$

Close bound (`close`) We call $O_v(x)$ the indicator function of the overlap between the bound interval of both children of node x:

$$O_v(x) = \mathbb{1}_{\min[u_0(x), u_1(x)] \geqslant \max[l_0(cx), l_1(x)]} \tag{5.38}$$

At node x, the probability θ_x to select child 1 is defined by

$$\theta_x = \begin{cases} 0.5 & \text{if } O_v(x) = 1 \\ \frac{l_1}{u_0 + l_1} & \text{if } O_v(x) = 0 \text{ and } u_0(x) < l_1(x) \\ \frac{u_1}{l_0 + u_1} & \text{if } O_v(x) = 0 \text{ and } u_0(x) \geqslant l_1(x) \end{cases} \tag{5.39}$$

The `close` bound is illustrated on figure 5.10. This strategy is less aggressive than the other two because as long as the bounds of both children overlap, the sampling is uniform. And when they no longer overlap, the bounds used to compute Φ are the closest to each other.

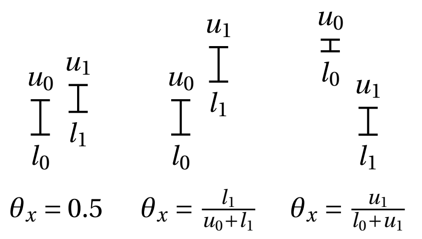


Figure 5.10 – Illustration of the `close` policy

SVM experiment on a synthetic problem

To get an idea of the behaviour of the IST policy for SVM evaluation with a Gaussian kernel, we use a synthetic synthetic problem, a 10×10 checkerboard. The data set, depicted on figure 5.11, is a 2-class problem made of 2D-Gaussian clouds of points centred at each integer coordinate (from 0 to 9) and of standard deviation 0.3 alternating classes.

Three parameters are involved in the IST SVM evaluation: C and γ, the usual SVM parameters and T, the number of samplings with replacement that are performed in the evaluation. These parameters were chosen by cross-validation to get a good validation accuracy and a small actual number of kernel computation (see table 5.3).

The results are illustrated on figure 5.12a. The `exact` method is the usual SVM policy (equation 5.20). For the sampling methods, we vary the number of samplings T to do the estimation of the SVM prediction from 10 to 100,000.

SVM experiment on real data

We applied this IST method to the Gaussian kernel SVM trained on the Covertype data set (Bache and Lichman). It contains 522,910 samples of dimension 54 for training and 58,102 for testing. As in Collobert et al. (2002) we consider the binary classification problem (class 2 versus 6 others). We carefully selected the parameters through cross-validation to train the model which in the end contains 105,492 support vectors. The right plot of figure 5.12 shows the test accuracy versus the average number of inner products performed during sampling. The exact computation of the SVM decision (see equation 5.20) yields an error rate of 1.765% and requires 105,492 inner products.

We have also tried the IST method on other data sets such as `webspam` or `a9a` but the computation cost of IST was greater than for the default policy (*i.e.* evaluating all the dot products),

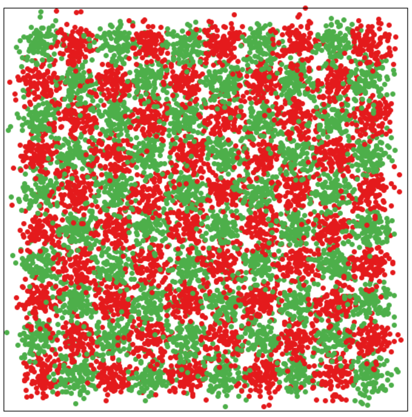
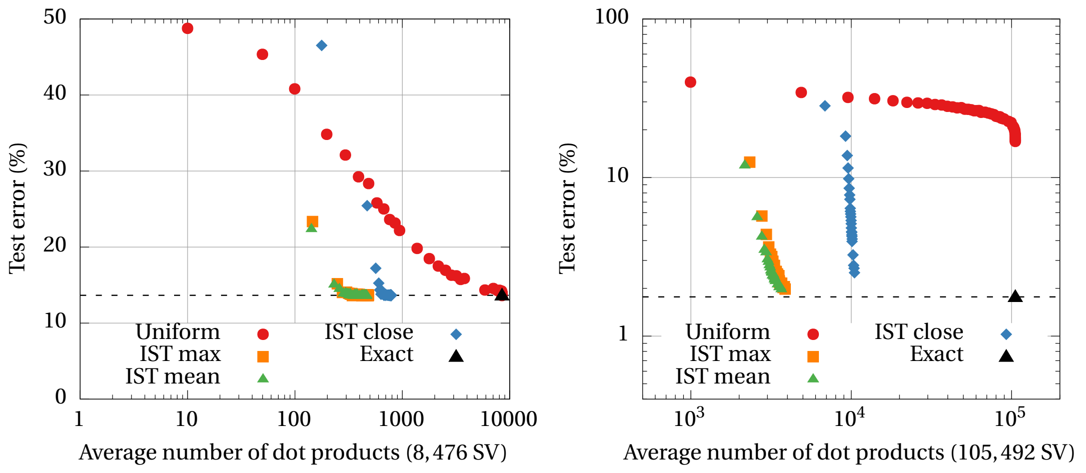


Figure 5.11 – Synthetic data set checkerboard



(a) Checker board (b) Covertype

Figure 5.12 – Test error vs. average number of inner products for the synthetic problem and for the Covertype data set (bottom). Each point corresponds to a fixed number of samplings to estimate the prediction value. The greater the number of samplings, the better the estimate. IST methods reach top accuracy with less inner product computations. For the Covertype data set (bottom), the exact point performs all the computations and reaches 1.765% error rate. For the sampling methods, the number of samplings is varied from 1,000 to 300,000. For this particular data set, the distribution of kernel responses is so skewed that IST requires very few inner products and reaches low error, while the uniform approach never reaches good estimates.

C	γ	T	Kernels	#SV	Acc.	Time (s)
0.1	50	100000	265.24	7981	83.95	130.08078
0.5	50	100000	229.703	6213	83.95	125.11008
0.5	15	100000	357.499	4480	84.15	122.91283
0.5	10	100000	437.615	4185	84.25	122.89537
1	5	100000	620.246	3691	84.45	122.06775
5	5	100000	559.196	3217	83.95	120.50157
0.1	20	20000	369.676	7220	84.55	26.14553
0.1	5	20000	766.475	5758	**84.75**	25.972519
0.1	10	20000	513.464	6203	**84.75**	25.817479
0.5	50	20000	**214.318**	6213	83.95	25.072426
0.5	10	20000	396.372	4185	84.25	24.709539
0.1	50	10000	238.635	7981	83.95	13.116903
0.5	20	10000	279.822	4788	83.95	12.406951
0.1	5	5000	673.641	5758	84.15	6.62012
0.1	20	5000	337.403	7220	84.55	6.594196
0.1	10	5000	459.707	6203	**84.75**	6.57555
2	20	5000	235.904	3836	83.95	6.076082
0.1	5	2000	610.77	5758	84.15	2.746622
0.1	20	2000	314.86	7220	84.55	2.691372
0.1	15	2000	353.546	6766	**84.75**	2.674636
0.1	5	1000	562.106	5758	**84.75**	1.451517
0.1	10	1000	395.427	6203	84.25	1.392765
0.1	20	1000	298.203	7220	84.35	1.377687
0.5	15	1000	264.911	4480	84.05	1.30869
0.1	20	500	280.077	7220	84.15	0.736825
0.1	15	500	310.606	6766	84.45	0.729109
0.1	10	200	326.765	6203	84.65	0.347478
0.1	15	200	281.025	6766	83.95	0.334733
0.1	20	100	234.961	7220	**84.75**	0.197346
0.1	15	50	227.658	6766	84.15	**0.128896**

Table 5.3 – Cross validation to choose parameters C and γ for the checkerboard data set. T can be reduced for speedup or increased for accuracy.

because not only IST requires to compute many of the inner products with the support vectors but also almost all of the inner products with the centroids which lead to twice as many computations.

The Covertype data set has this property that at test time, only very few support vectors matter in the decision ($\simeq$100 out of the 100,000 support vectors) which makes the clustering and the sampling very efficient. As demonstrated by Jose et al. (2013) or Hsieh et al. (2014), this data set is prone to huge gain in prediction.

5.5 Discussion

The Importance Sampling Tree introduced in this chapter is designed to improve the learning of a classifier by allowing to sample important points, and at the same time, by providing a compensation for the induced bias.

The IST is a binary tree built on the training data in such a way that it reflects the statistical regularities of the samples. We have shown that a simple top-down 2-means partitioning was sufficient to organise the data. The other important component of the IST structure is the ability to provide unbiased estimates of the true weights below a node, although the sampling is not performed in a uniform manner.

In its current formulation, the IST accumulates all statistics that are observed. This can be a weakness of the method so far, because in this situation, early observations are still taken into account, although the classifier has changed. Some samples might be difficult in the first place because of the sampling, and then quickly become easy, and yet, their initial weights are still aggregated. A first way for improving the IST is to use a moving average, so that late weights have more influence. We could also imagine to anticipate the future values of the weights, based on some probe samples.

The second point is the sampling procedure. We have proposed two strategies, one using empirical weight estimates, the other exact bounds on the said weights. The latter is exact but can be used only with a strong prior information about these bounds (*e.g.* geometrical information consistent with the weight distribution), which is not the case for very large training sets. What is missing in our approach is a bandit-type use of a confidence interval on the estimate. For instance based on the Hoeffding's inequality, to get something similar to the UCB policy used in MCTS.

6 Active Sampling Tree for Support Vector Machines

Contents

This chapter presents a simple yet efficient framework, called an "Active Sampling Tree" to train more efficiently online *support vector machine classifiers. The method builds upon the Active Selection of LASVM introduced by Bordes et al. (2005), and makes use of a sampling tree to mimic the Active Selection.*

$f(x)$	Response of the SVM classifier on sample x
$\mathcal{S}$	Set of support vectors indices in the training set
L	The number of leaves/samples in the Active Sampling tree
N	A random variable on $[\![1, L]\!]$ returning an image index in the training set
M	Number of candidate samples in the Active Selection strategy
$\Psi(\alpha)$	Model of the distribution $P_{N \sim \mathcal{U}}(\lvert f(x_N)\rvert \leqslant \alpha)$
U	Number of additional leaf weight to update in the Active Sampling Tree
$w(f(x))$	A function of compute a sampling weight from a classifier response

Table 6.1 – Notations

6.1 Introduction

This chapter is in the continuation of the previous one, but is more specific to support vector machines. We have introduced the previous chapter by showing that, when training a classifier, the careful selection of training samples can be important, or even the modulation of their contribution in the training loss: recall that Fleuret and Geman (2008) as well as Kalal et al. (2008) showed improved performance for boosting when the training samples are selected proportionally to their boosting weight, while Zhao and Zhang (2014) and Needell et al. (2013) showed a reduction of the variance of the gradient estimate when using importance sampling to make the mini batch. Moreover, as demonstrated in the previous chapter, selecting training samples based on their boosting weights or their gradient norm reduces the training loss faster and accelerates the convergence.

Here, we are interested in the Active Selection for the LASVM solver introduced by Bordes et al. (2005) (described in detail in 6.2.2). Bordes et al. showed that training an online SVM by selecting samples close to the current boundary results in more compact models with fewer support vectors, while reaching equivalent or superior accuracy compared to standard algorithms.

The Active Selection first requires to select M samples uniformly in the training set (usually 50), to evaluate the classifier on them, and then to choose the one closest to the boundary to train. For one sample used by the learner, M are actually looked at, which is the bottleneck of the method.

In this chapter, we introduce an Active Sampling Tree, which is designed to emulate the Active Selection by direct sampling, without the rejection phase. It is achieved by first organising the training samples with a tree similar to the one used in the Importance Sampling Tree presented in the previous chapter, and second by using a sampling strategy specifically designed to mimic the probability distribution as the Active Selection.

Before describing more precisely the Active Sampling Tree, the following section describes the LASVM algorithm and the Active Selection introduced by Bordes et al. (2005). Table 6.1 contains the various notations used in this chapter.

6.2 Related work

6.2.1 Support Vector Machines

Support vector machines (SVM) were introduced by Vapnik and Lerner (1963) and the current formulation of the problem was presented by Cortes and Vapnik (1995). SVM have been very popular in many communities because there are efficient and give good accuracy even with a small number of training samples. Fernández-Delgado et al. (2014) compared 179 classifiers on 121 data sets, and showed that SVM (with a Gaussian kernel) along with random forests are the among the best machine learning algorithms achieving the highest accuracy.

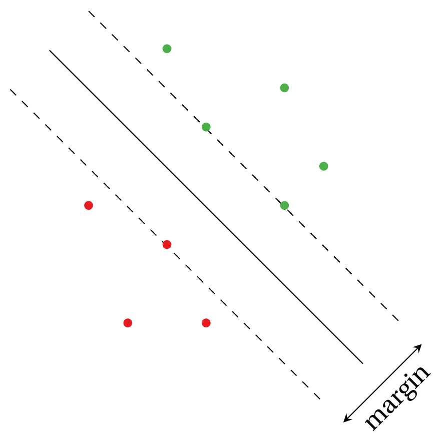


Figure 6.1 – Support vector machines find the optimal margin to separate the two classes

Support vector machines for classification are supervised learning algorithms which aim at finding the optimal hyperplane to separate two classes. Given a labelled training set $(x_n, y_n) \in \mathbb{R}^D \times \{-1, +1\}$, the SVM aims at separating the binary data by maximising the margin between the two classes (see figure 6.1) and solves the following optimisation problem:

$$\min_{w,b,\xi} \frac{1}{2}||w||^2 + C\sum_{n=1}^{N} \xi_n \qquad \text{s.t.} \quad \begin{cases} y_n(w^T\phi(x_n) + b) \geqslant 1 - \xi_n \\ \xi_n \geqslant 0 \end{cases} \tag{6.1}$$

where $C > 0$ is a penalty constant, w the hyperplane separating the two classes, ϕ a function that maps x in a higher dimensional feature space and ξ a relaxing variable that allows some training points to violate the margin when the training set is not separable.

In practice, the problem is usually solved in the dual space by solving

$$\max_{\alpha} \sum_n \alpha_n y_n - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \quad \text{s. t.} \quad \begin{cases} \sum_n \alpha_n = 0 \\ A_n \leqslant \alpha_n \leqslant B_n \\ A_n = \min(0, C y_n) \\ B_n = \max(0, C y_n) \end{cases} \tag{6.2}$$

where K is the kernel function between two samples.

There are various ways to solve this optimisation problem. The LIBSVM package by Chang and Lin (2011) uses the sequential minimal optimisation algorithm (Platt, 1998), the LIBLINEAR package by Fan et al. (2008) uses a dual coordinate descent for the linear case, and SVM^{light} use a decomposition strategy of the training samples and a selection of the variables to update. These solvers are batch methods which require the training samples to fit in memory which becomes impractical with large data sets.

To overcome the issue of training SVM on large data sets, the online approach can be considered. In this setup, training samples are used one at a time or in mini batches, and such strategies aim at converging to the same optimal solution. The Pegasos algorithm by Shalev-Shwartz et al. (2011) derives a stochastic sub-gradient descent policy with a carefully chosen step size. The other online SVM solver is LASVM by Bordes et al. (2005) that we now present in more detail. The online setting has the advantage of letting a policy to carefully select the next samples to feed to the learner, and this is what we are interested in here.

6.2.2 LASVM and Active Selection

LASVM was introduced by Bordes et al. (2005) as an online SVM solver which converges to the true SVM solution and can already yield competitive accuracy after a single pass through the whole training set. Algorithms 3, 4 and 5 are the three main components of LASVM.

To present the LASVM algorithm, we use the same notations as the original authors: we call $\mathcal{S}$, the set of support vectors indices; LASVM will sequentially fill and update $\mathcal{S}$. At a given time t, the response of the current classifier on sample x is

$$f(x) = \sum_{s \in \mathcal{S}} \alpha_s K(x, x_s) + b_t, \tag{6.3}$$

where b_t is the bias of the classifier at time t.

For a start, LASVM populates $\mathcal{S}$ with random indices (5 of each class in practice). After the algorithm has been initialised, the algorithm visits a predefined number of samples, possibly all the training samples, and possibly over multiple epochs, in the following way (algorithm 3): first, the policy selects a non visited sample $i \notin \mathcal{S}$ and evaluates if it can be a potential support vector (the process step, algorithm 4). If the sample is added to the cache of support vectors, the Lagrange coefficients α_n are updated accordingly. Then, the algorithm removes from the cache some samples which are no longer support vectors (the reprocess step, algorithm 5). This latter operation can be repeated multiple times per process operation. Finally, after the predefined number of iterations, the reprocess step is repeated until convergence of the solution.

As the samples are visited one at a time, Bordes et al. have investigated several strategies to select the next sample to use in the online setting and their influence on the convergence and

Algorithm 3 LASVM (Bordes et al., 2005, section 3.2)

```
1: Seed $\mathcal{S}$ with a few samples of each class            ▷ Initialisation
2: $\alpha \leftarrow 0$ and compute the initial gradient $g$
3: **repeat**                                                       ▷ Online Learning
4:     Select a sample $k_t$
5:     PROCESS($k_t$)
6:     REPROCESS() once
7: **until** a predefined number of iterations
8: Call REPROCESS() until $\delta \leqslant \tau$                    ▷ Finishing step
```

Algorithm 4 LASVM Process(k) (Bordes et al., 2005, section 3.2)

```
1: Bail out if $k \in \mathcal{S}$
2: $\alpha_k \leftarrow 0, \quad g_k \leftarrow y_k - \sum_{s\in\mathcal{S}} \alpha_s K_{ks}, \quad \mathcal{S} \leftarrow \mathcal{S} \cup \{k\}$
3: **if** $y_k = +1$ **then**
4:     $i \leftarrow k$, $j \leftarrow \operatorname{argmin}_{s\in\mathcal{S}} g_s$ with $\alpha_s > A_s$
5: **else**
6:     $j \leftarrow k$, $i \leftarrow \operatorname{argmax}_{s\in\mathcal{S}} g_s$ with $\alpha_s < B_s$
7: **end if**
8: Bail out if $(i, j)$ is not a $\tau$-violating pair
9: $\lambda \leftarrow \min\left\{\frac{g_i - g_j}{K_{ii}+K_{jj}-2K_{ij}}, B_i - \alpha_i, \alpha_j - A_j\right\}$
10: $\alpha_i \leftarrow \alpha_i + \lambda, \quad \alpha_j \leftarrow \alpha_j - \lambda,$
11: $\forall s \in \mathcal{S}, \quad g_s \leftarrow g_s - \lambda(K_{is} - K_{js})$
```

Algorithm 5 LASVM Reprocess() (Bordes et al., 2005, section 3.2)

```
1: $i \leftarrow \operatorname{argmax}_{s\in\mathcal{S}} g_s$ with $\alpha_s < B_s$
2: $j \leftarrow \operatorname{argmin}_{s\in\mathcal{S}} g_s$ with $\alpha_s > B_s$
3: Bail out if $(i, j)$ is not a $\tau$-violating pair
4: $\lambda \leftarrow \min\left\{\frac{g_i - g_j}{K_{ii}+K_{jj}-2K_{ij}}, B_i - \alpha_i, \alpha_j - A_j\right\}$
5: $\alpha_i \leftarrow \alpha_i + \lambda, \quad \alpha_j \leftarrow \alpha_j - \lambda,$
6: $\forall s \in \mathcal{S}, \quad g_s \leftarrow g_s - \lambda(K_{is} - K_{js})$
7: $i \leftarrow \operatorname{argmax}_{s\in\mathcal{S}} g_s$ with $\alpha_s < B_s$
8: $j \leftarrow \operatorname{argmin}_{s\in\mathcal{S}} g_s$ with $\alpha_s > B_s$
9: **for all** $s \in \mathcal{S}$ s.t. $\alpha_s = 0$ **do**
10:     **if** $y_s = -1$ and $g_s \geqslant g_i$ **then** $\mathcal{S} = \mathcal{S} - \{s\}$ **end if**
11:     **if** $y_s = +1$ and $g_s \leqslant g_j$ **then** $\mathcal{S} = \mathcal{S} - \{s\}$ **end if**
12: **end for**
13: $b \leftarrow (g_i + g_j)/2, \quad \delta \leftarrow g_i - g_j$
```

the final performance. They have shown that by carefully selecting the next sample to process, the size of the solution (*i.e.* the number of support vectors) can be drastically reduced while achieving better test accuracy. The possible selection strategies are the following:

Uniform selection (or random selection) selects uniformly a sample among the non visited ones.

Gradient selection selects the worst classified sample among M uniformly selected non visited samples (*i.e.* $\operatorname{argmin}_n y_n f(x_n)$).

Active selection selects the closest sample to the current boundary among M uniformly selected non visited samples (*i.e.* $\operatorname{argmin}_n |f(x_n)|$).

Auto Active selection is a variant of the Active selection in which the size of the selected samples is adapted on the fly. It selects M non visited samples but stops as soon as 5 of them fall inside the margin (*i.e.* $|f(x_n)| \leqslant 1$), and then selects the closest to the margin.

Bordes et al. showed that the active selections were better than the uniform selection and the gradient selection, and Loosli et al. (2007), have confirmed these observations when training a digit classifier with artificial distortions. The active selections yield a better test accuracy with a smaller number of support vectors. This result may sound surprising because we would expect to obtain a better model by selecting highly misclassified samples, or high gradient samples, which corresponds to selecting difficult samples. In fact, selecting samples close to the boundary (the active selections) prevents outliers to be added in the final model (when not all training samples are used), which causes the model to be more compact, faster, and more accurate.

During our experiments, we have observed a behaviour of the active selections that was not reported by Bordes et al. and which is depicted on figure 6.8 (curves “Uniform” and “Active Selection”). The charts were plotted by dumping the model at regular intervals and by evaluating it on the test set: since the learning is online, it can be interrupted anytime. When training LASVM with one of the active selections on a finite training set (here on the Income Census data set), the learning quickly reaches a better model than the uniform approach with fewer support vectors, but later in the process, when the outliers start being visited (and are therefore added to the model), the accuracy of the model declines very fast, and when all the samples are visited, the accuracy is worse than with the uniform selection. This is because the model starts being “polluted” with difficult samples that are not representative of their class. In the next epoch (not shown on figure 6.8), the samples that were removed or not inserted in the previous epoch become support vectors, and the accuracy of the model improves, and finally reaches the same accuracy as the baseline. We therefore insist on the fact that the active selections are very good strategies only if not all the samples are visited.

As mentioned above, the Active Selection visits M samples before processing only one. The next chapter describes our Active Sampling Tree, which aims at emulating the Active Selection by *direct* sampling, with a view to being faster than the original method.

6.3 Active Sampling Tree

The main idea of the Active Sampling Tree (AST) we are introducing here, is to emulate the Active Selection described in section 6.2.2 by a direct sampling policy. By *direct*, we mean that we would like to avoid looking at M samples before choosing the one to process (what we could call *indirect* sampling), and we would like to apply a policy that would directly choose the sample to process.

We have presented in section 5.2.3 several ways to sample according to a discrete distribution, and we have shown that the tree-based sampling (the roulette wheel selection) was particularly well suited for both efficient sampling and updating.

In the same spirit as our Importance Sampling Tree, we use a binary tree whose leaves carry the training samples and their weight used for sampling. The next section shows what should be the weights at the leaves, so that the roulette wheel selection mimics the Active Selection process.

6.3.1 The Active Sampling Tree weight

The Active Selection process, which first randomly selects M samples uniformly in the set of non processed samples, and then selects the one closest to the boundary, is a random process, which therefore has a probability distribution. In the previous section, we have presented various ways to sample from a discrete distribution, and we would like to emulate the Active Selection by sampling. The probability distribution characterising the Active Selection is constantly changing because of the online setting which modifies the classifier at each iteration, and so the best choice to make is the tree structure to perform a discrete sampling, and to update it efficiently. The problem now boils down to finding the proper weight of the leaves to actually emulate the Active Selection.

In the sequel, we derive the mathematical form of the probability density of the Active Selection and deduce the proper tree weight.

Given a training set of L samples, the Active Selection uniformly first selects M candidate indices $N_1, \ldots, N_M \in [\![1, N]\!]$, and among them selects the one that minimises the distance to the boundary $|f(.)|$. Let us call $N^\star$ the selected one. We have:

$$\begin{aligned}
\mathbb{P}(N^\star = n) = &\sum_{k=1}^{M} \mathbb{P}(N^\star = n \mid N_k = n)\mathbb{P}(N_k = n) \\
&+ \underbrace{\mathbb{P}(N^\star = n \mid \forall\, m, N_m \neq n)}_{=0}\mathbb{P}(\forall\, m, N_m \neq n) && (6.4)\\
= &\sum_{k=1}^{M} \mathbb{P}(\forall\, l \neq k, |f(x_{N_l})| > |f(x_{N_k})| \mid N_k = n)\, \mathbb{P}(N_k = n) && (6.5)\\
= &\sum_{k=1}^{M} \mathbb{P}(\forall\, l \neq k, |f(x_{N_l})| > |f(x_n)|)\, \mathbb{P}(N_k = n) && (6.6)\\
= &\sum_{k=1}^{M} \left[\prod_{l \neq k} \mathbb{P}(|f(x_{N_l})| > |f(x_n)|)\right] \mathbb{P}(N_k = n) && (6.7)\\
= &\sum_{k=1}^{M} \mathbb{P}(|f(x_{N_l})| > |f(x_n)|)^{M-1}\, \mathbb{P}(N_k = n) && (6.8)\\
= &\ \mathbb{P}(|f(x_N)| > |f(x_n)|)^{M-1}\, [1 - \mathbb{P}(\forall\, m, N_m \neq n)] && (6.9)\\
\mathbb{P}(N^\star = n) \propto &\ \mathbb{P}(|f(x_N)| > |f(x_n)|)^{M-1} && (6.10)
\end{aligned}$$

If we have a model $\Psi(\alpha)$ for $\mathbb{P}_{N \sim \mathcal{U}}(|f(x_N)| \leqslant \alpha)$, then the weight of leaf n is

$$w_n = \left[1 - \Psi(|f(x_n)|)\right]^{M-1} \propto \mathbb{P}(N^\star = n). \tag{6.11}$$

Therefore, if the L samples are at the leaves of a binary tree with weight

$$w(f(x)) = [1 - \Psi(|f(x)|)]^{M-1}, \tag{6.12}$$

then sampling this tree with the roulette wheel selection policy described in 5.2.3 will result in sampling according to the Active Selection. Note that the weights need not be normalised.

As a sanity check, figure 6.2 shows on two simple examples that the AST indeed emulates the Active Selection. We generate L values (*i.e.* the $f(x_n)$) and perform the Active Selection and the AST, and compare the two resulting empirical distributions. For the Active Sampling, we set the leaves with their true weights obtained from equation 6.11. As depicted on figure 6.2, the two empirical distributions are the same.

The main problem with sampling to emulate the Active Selection is that it requires to know all the weights to perform the sampling, which in practice would imply to compute the response of the current classifier on all the non processed samples. The next section presents how this can be avoided, and how the samples can be organised once for all so as to get information of other samples than the one begin processed.

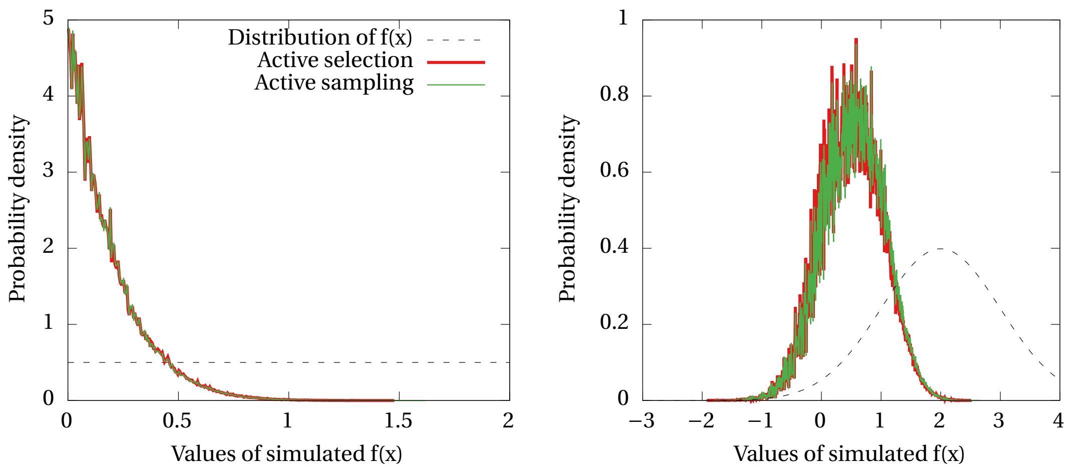


Figure 6.2 – Comparison of the Active Selection and Sampling for $M = 10$ candidates, where $f(x)$ follows a uniform distribution in $[0,2]$ (left) or a Gaussian distribution $\mathcal{N}(2,1)$ (right). As expected, the distribution resulting from our Active Sampling model follows closely the empirical distribution observed under the Active Selection policy.

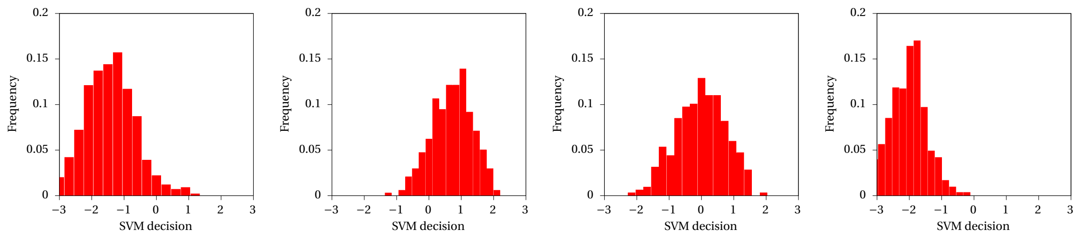


Figure 6.3 – Histogram of SVM responses on samples belonging to the same cluster. This experiment was done on the Census Income data set. The samples were first top-down clustered, and then a SVM classifier was applied on samples belonging to the same cluster (*i.e.* here, samples sharing a common ancestor in the tree.)

6.3.2 Building the Active Sampling Tree

As for the Importance Sampling Tree of chapter 5, the AST is only valid if the weights of the samples/leaves are known, meaning that the classifier has to be evaluated on the samples (to get $|f(x)|$ in equation 6.11), which is costly.

And as for boosting or neural networks, a SVM classifier has correlated responses on similar points (*i.e.* with small Euclidean distance). This is due to the fact that the most commonly used kernels (linear, polynomial, Gaussian) are based on the Euclidean distance. To illustrate this, we have clustered a training set with a simple k-means algorithm, then applied a SVM classifier on the samples, and plotted a histogram of the responses of samples belonging to the same cluster. Figure 6.3 shows the result of this experiment, which demonstrates that samples from the same cluster follow a quite peaked Gaussian distribution.

If we want to use a sampling tree without all its leaf weights to be known, one way would be to use a tree of samples which reflects the statistical regularities of the training set: we want similar training samples to be in nearby leaves (*i.e.* small path between leaves). As we showed that a simple clustering already gives consistency, we follow the same recipe as in the previous chapter to build the tree: a vanilla 2-means clustering is recursively applied on the samples until reaching two samples. Now, if a leaf is sampled, then it is also informative on the leaves nearby, and updating the leaf with the corresponding weight will add consistency in the weights above.

6.3.3 The Active Sampling Tree policy

Algorithm 6 presents the main workflow to train LASVM with the Active Sampling Tree.

The tree weights are initialised to 0 at the beginning. The classifier is first initialised with U_0 random training points. The corresponding weights of the tree are updated accordingly with $w(f_t(x_t))$, where x_t is the sample selected at iteration t and f_t is the classifier obtained after t iterations.

After this initialisation, each step of AST consists of an update step in which we update the weights of U uniformly selected training points, and of a sampling step in which we sample a point according to the current weights of the tree. Then we train the classifier with that sample, and we can also update its own weight. At each iteration, the tree is updated with $1+U$ fresh weights.

Note that contrary to the IST (chapter 5), we sample without replacement (our tree forbids to revisit an already visited leaf) and we do not compensate for the bias that is induced by the policy. The Active Selection of Bordes et al. (2005) does not compensate for this selection either so AST does not. With the IST, samples can be revisited many times and so the contribution of a sample is divided by its probability of being sampled.

6.3.4 The model for Ψ

Given equation 6.12, a model is required for

$$\Psi(\alpha) = \mathbb{P}_{N\sim\mathcal{U}}(|f(x_N)| \leqslant \alpha). \tag{6.13}$$

Ψ describes the probability that a sample x_n has a response smaller than $\alpha > 0$, which can then be converted into a sampling weight to emulate the Active Selection.

In practice, one observes that the responses of the classifier over the training samples follow a Gaussian distribution with mean 1 and of standard deviation around 1. We can then compute

Algorithm 6 Training LASVM with the Active Sampling Tree

```
1:  $\mathcal{T}$, a tree reflecting the regularity of the data set with the training points at the leaves
2:  $\Psi$, a model of the responses of the classifier
3:  $T$, number of training iterations
4:  Train $f$ with $U_0$ random training points
5:  for $t = U_0$ to $T$ do
6:      for $u = 1$ to $U$ do
7:          Sample uniformly a leaf/sample $s$ of $\mathcal{T}$,
8:          Compute the responses of the current classifier $f_{t-1}(s)$,
9:          Update $\mathcal{T}$ and $\Psi$ accordingly.
10:     end for
11:     Sample a leaf/sample $s$ with AST
12:     Train $f_{t-1}$ with $s$ to get $f_t$
13: end for
```

the exact weight under a Gaussian model. We call $\mathcal{G}(x,\mu,\sigma)$ the Gaussian distribution.

$$\begin{aligned}
\Psi(\alpha) &= \mathbb{P}_{N\sim\mathcal{U}}(|f(x_N)| \leqslant \alpha)\\
&= \mathbb{P}(-\alpha \leqslant f(x_N) \leqslant \alpha)\\
&= \int_{-\alpha}^{\alpha} \mathcal{G}(x,\mu,\sigma)dx\\
&= \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\alpha}^{\alpha} \exp\left(-\left(\frac{x-\mu}{\sqrt{2}\sigma}\right)^2\right)dx\\
&= \int_{\frac{-\alpha-\mu}{\sqrt{2}\sigma}}^{\frac{\alpha-\mu}{\sqrt{2}\sigma}} \frac{1}{\sqrt{2}\sigma}\frac{1}{\pi} e^{-u^2}\sqrt{2}\sigma du, && \text{with } u = \frac{x-\mu}{\sqrt{2}\sigma} \text{ and } du = \frac{dx}{\sqrt{2}\sigma}\\
&= \frac{1}{\pi}\int_A^B e^{-u^2} du && \text{with } A = \frac{-\alpha-\mu}{\sqrt{2}\sigma} \text{ and } B = \frac{\alpha-\mu}{\sqrt{2}\sigma}\\
\Psi(\alpha) &= \frac{1}{2}\left[\operatorname{erf}(B) - \operatorname{erf}(A)\right] && (6.14)
\end{aligned}$$

The weight of a leaf can be computed with equations (6.12) and (6.14), and by keeping an estimate of the the mean μ and the standard deviation σ of the Gaussian model.

6.4 Experiments

This section presents the behaviour of the AST tree when training a LASVM classifier compared to the uniform selection and the Active Selection of training samples. We apply the AST on a synthetic 2D example to visualise the samplings, on a popular SVM data set and on two large scale image data sets. The parameters used are summarised in table 6.2.

We will mainly focus on the accuracy as a performance criterion and we will look in detail at how fast the accuracy is increasing. In particular, we will monitor the number of samples pro-

Data set	Dimension	C	γ
Banana	2	1	1
InfiMNIST	784	64	0.03
CIFAR10	800	10	0.0009
Census Income	123	300	0.0003

Table 6.2 – Parameters of the Gaussian kernel for the data sets

cessed by LASVM disregarding how many other samples were actually looked at, and possibly discarded (axis “Samples processed” in the following charts), as well as the computational cost of the various selections. Indeed, while the uniform selection processes all selected samples, the Active Selection first selects M of them requiring computing possibly $M \times S$ kernel values, where S is the number of support vectors currently in the cache.

6.4.1 Training Image classifiers

We recall here that an image classifier is trained with a collection of labelled images and the more the training data, the better the final performance. However, labelled data is usually expensive to get resource-wise, and data sets are usually augmented with artificial distortions of the original images, such as translations, rotations, scaling or elastic distortions. The assumption is that a distorted image still belongs to the same original class. An original data set of a few thousand images can therefore yield a data set of several million images.

6.4.2 Building a tree of distortions

As described in section 6.3.2, we build a tree that reflects the regularities of the image data set. To avoid performing the recursive top-down 2-means clustering on millions of images, we only perform the clustering on the original images, by first splitting the classes and then extending the tree at the bottom by adding the corresponding distortions of each original. This clustering has been shown to be consistent with the sampling framework as in chapter 5.

6.4.3 Experiments on a synthetic data set

To get a visual understanding of the behaviour of the AST compared to the uniform selection and the Active Selection, we use the synthetic banana data set. It is a binary problem of 4,000 2D training points with a banana shape for which the Gaussian kernel is well suited. On figure 6.4, we visualise the first 1,000 training points that are selected by the three policies.

The uniform policy gives an overview of the full training set (figure 6.4a). The Active Selection only considers points that lie at the boundary (figure 6.4b) after evaluating the current classifier $M = 50$ times on points sampled uniformly.

The AST is less aggressive than the Active Selection but properly concentrates at the boundary

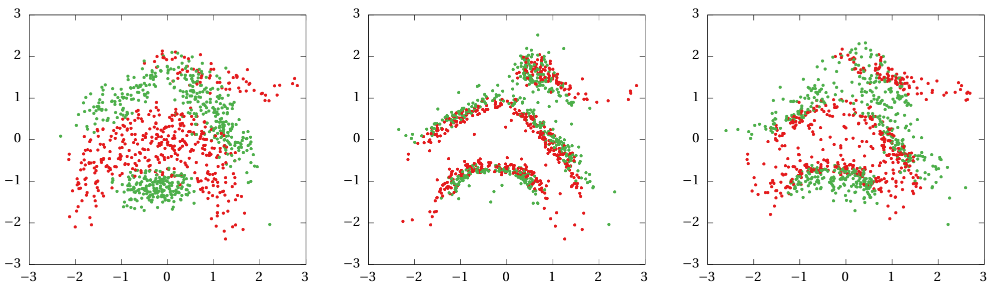


Figure 6.4 – These plots show which samples were selection by the uniform method, by the Active Selection, and by the Active Sampling Tree.

of the classifier. For this experiment, the classifier was initialized with 20 random points, and then the AST policy was used after updating $U = 10$ random leaves at each iteration. This experiment shows that the AST is doing what it is supposed to do, and concentrates at the boundary of the classifier.

6.4.4 Experiments on InfiMNIST

Following the work of Loosli et al. (2007), we apply the Active Sampling Tree to train a digit classifier on the InfiMNIST data set. We presented this data set in chapter 2, but briefly recall it here. MNIST is a collection of 60,000 28 × 28 gray scale images of digit (from 0 to 9) and was extended by Loosli et al. (2007) by using sub-pixel transformations as described by Simard et al. (1992) and translations of 1 pixel. Although the resulting data set may contain several trillion images, we only use the first 8.1 million as in Loosli et al. (2007).

We train ten binary classifiers in a 1-vs-all fashion and for testing, we simply merge the classifiers and take the maximum response out of the ten. The features used are directly the image pixels which yields a dimension of 784. As a preprocessing step, the pixels are scaled to $[0,1]$ by simply dividing by 255. The parameters of the Gaussian kernel are $C = 64$ and $\gamma = 0.03$ and were determined by cross-validation with LIBSVM. Since this procedure is quite time-consuming, we have carried it out on the original MNIST data set because it would have been impossible to perform it on the 8.1M images. We also performed a cross-validation on 300,000 images (the original as well as four distortions per digit) and obtain roughly the same C and γ.

As for the Importance Sampling Tree of chapter 5, the data set is organised at the beginning of the training once for all. The original samples are top-down clustered, by first separating classes, and then recursively applying a 2-means algorithm. After this clustering, a leaf node corresponding to one image is replaced by a sub-tree containing several distortions as well as the original. The tree thus contains 8.1M leaves. Figure 6.5 shows a small scale example of tree with original MNIST images.

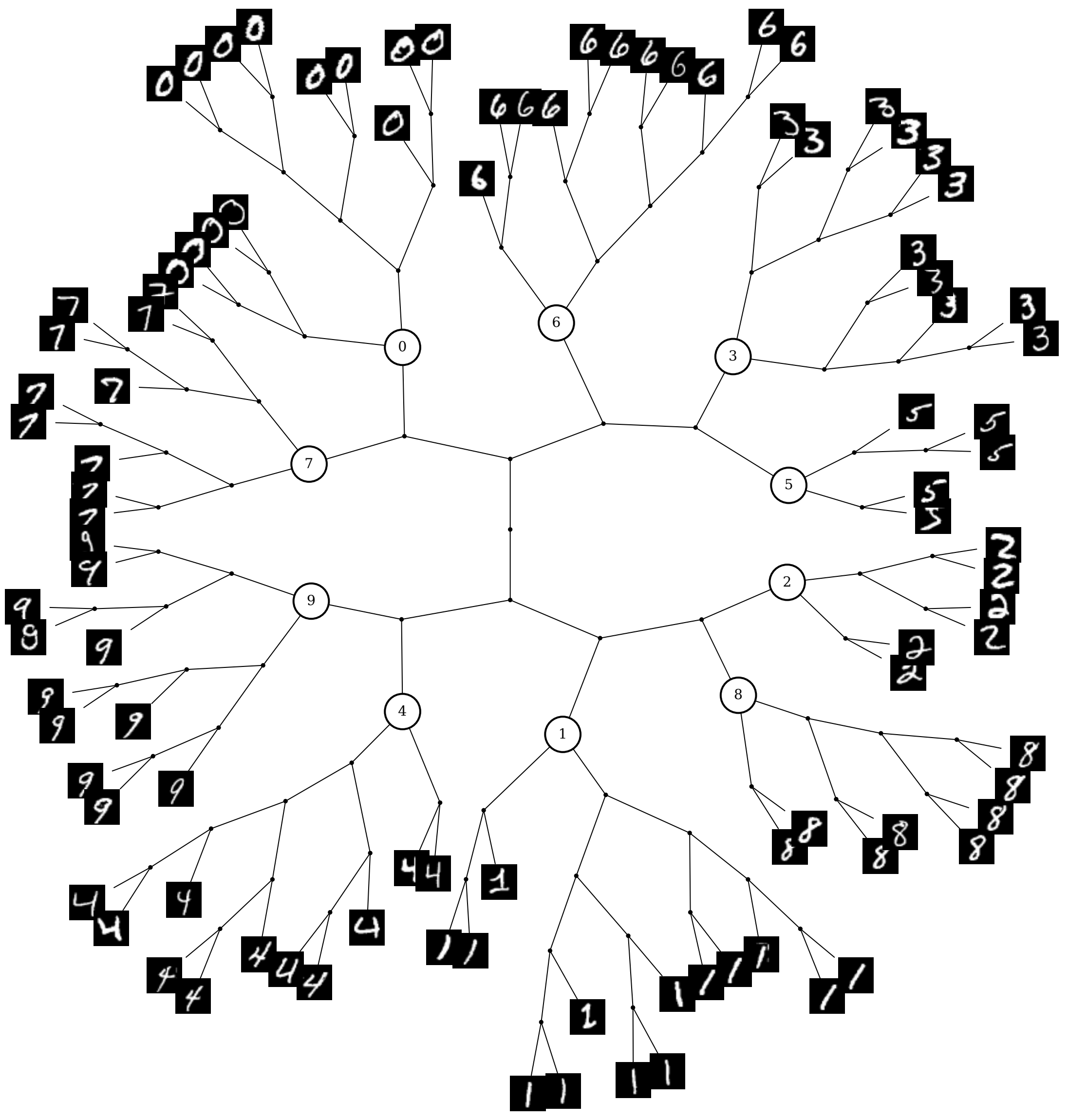


Figure 6.5 – Example of top-down 2-means clustering of MNIST original images. On this particular example, we can see in class 1 or class 4 how similar samples are properly grouped together (rotated 1 for instance). No additional distortion has been added in this example.

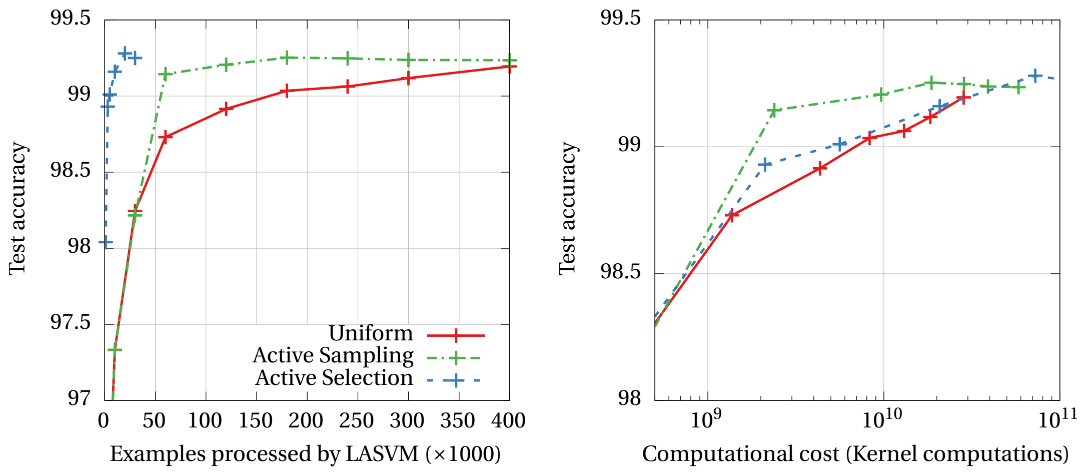


Figure 6.6 – Comparison of the uniform selection, Active Selection and the Active Sampling on the InfiMNIST data set. The Active Sampling was initialised with 30,000 original training samples before using AST and no extra leaf is updated ($U = 0$).

The training goes as follows. The first $U_0 = 30{,}000$ samples are uniformly selected in the original training set (no distortion). The weights of the selected leaves are updated accordingly. After this initialisation, the AST policy is used. As the InfiMNIST data set is quite smooth and not noisy, there is no need to update additional leaf weights ($U = 0$). The AST policy simply boils down to sampling a leaf and updating its weight.

Figure 6.6 shows the test accuracy as a function of the number of iterations and of the computations. The uniform selection reaches an accuracy of 98.7% after visiting 60,000 samples and constantly increases until reaching an accuracy of 99.2% after 400,000 samples. The Active Selection (with $M = 50$) reaches an accuracy of 99.28% after processing 20,000 samples (*i.e.* after visiting 1,000,000 samples, fifty times more) which results in fairly the same computational cost with respect to the accuracy. Regarding the accuracy as a function of the iterations (left plot), the Active Selection is clearly a good way to select informative samples. The AST does a good job in finding good samples (left plot): it reaches an accuracy of 99.14% after 60,000 points and 99.25% after 200,000 points.

6.4.5 Experiments on extended CIFAR10

We briefly recall that CIFAR10 is a collection of 50,000 32×32 colour images of 10 classes such as “bird”, “car”, or “truck”. We have extended this data set with mirrored version of the original images as well as translations of one pixel, yielding a new collection of 900,000 images.

The data set is organised by first splitting the data set by classes, and then by performing a top-down clustering the original images and by then extending the tree with the distortions of each sample.

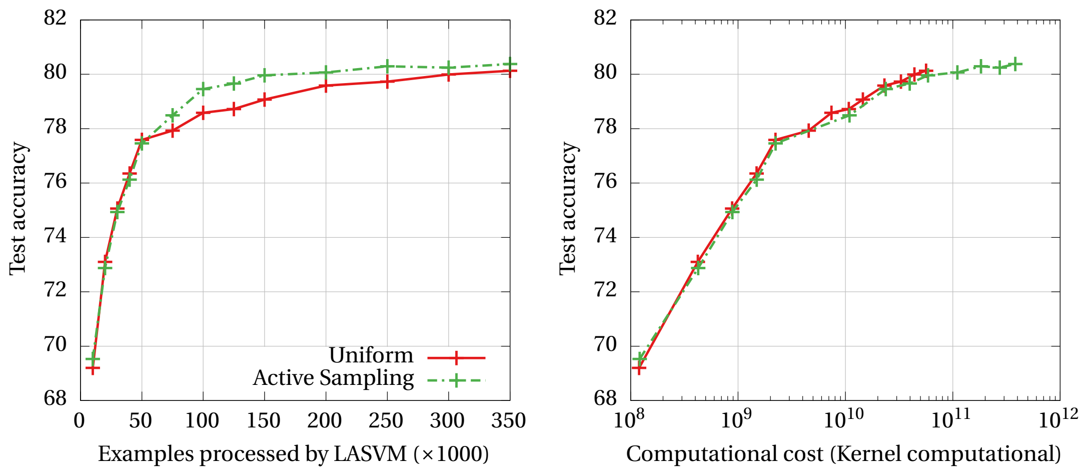


Figure 6.7 – Comparison of the uniform selection and the Active Sampling Tree on the extended CIFAR10 data set. AST was first initialised with 50,000 original training samples and each step of AST update $U = 2$ samples.

We use the features of Coates and Ng (2011) which we described in section 2.1.3. Each 6×6 image patch is projected on a dictionary of visual words containing 200 elements and the activations are pooled on a 2×2 grid, yielding a feature vector of dimension 800. The parameters of Gaussian kernel are $C = 10$ and $\gamma = 0.0009$ and were determined by cross-validation on the original CIFAR10 samples. We recall here that in the work of Coates and Ng (2011) and in the experiments we carried out in chapter 2, the classifier was linear and the dimension of the features was 12,800 (dictionary of size 3200 codes).

As the CIFAR10 data set is more noisy than the MNIST data set, it was necessary to update more leaves than just the one visited. We chose $U = 2$ in this experiment.

Figure 6.7 shows the results for this data set. The Active Selection could not be used here because it was too time consuming and required too much memory: in the previous experiment on InfiMNIST, the image features were the pixels themselves and the kernels could directly be computed from the pixel values. In this case, the single-layer features are expensive to compute (extraction of all patches, whitening, pooling, etc. see 2.1.3). The Active Selection therefore requires to compute M single-layer features to process one samples which is prohibitive.

On the extended CIFAR10 data set, the Active Sampling Tree reaches higher accuracy (left plot) for the almost the same computational cost (right plot). The extra computational cost $U = 2$ may hurt the cost of the AST while still appears to be necessary to prevent the tree from degenerating.

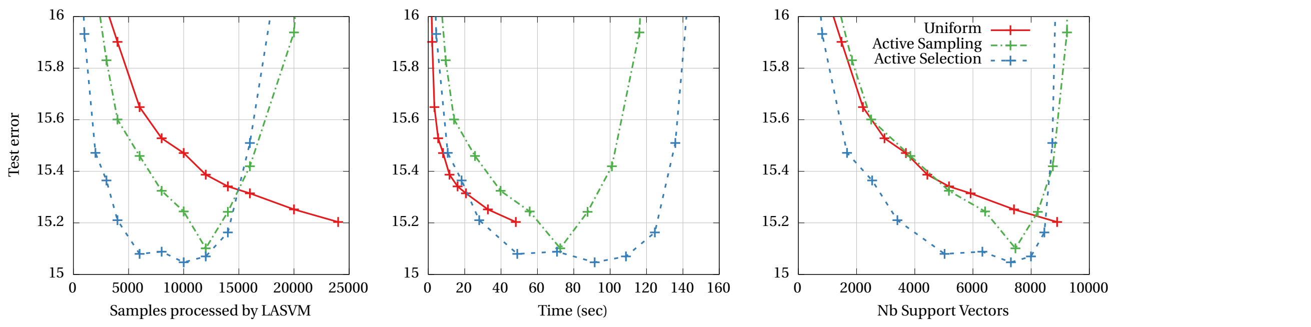


Figure 6.8 – Comparison of the selection strategies on the Census Income data set. The three graphs represent the test error as a function of the number of samples processed (left), of the training time (centre) and the number of support vectors (right). Experiments were averaged over 50 runs.

6.4.6 Experiments on the Census Income data set

We now compare the methods on the Census Income data set (also known as the Adult data set). The purpose is to predict whether the income of a person exceed $50,000 a year. It contains 26,008 unique training points of dimension 123. Figure 6.8 presents the results for the data set.

The left plot shows the test error as examples are processed by LASVM. The active policies select "better" samples than the uniform baseline because as the number of iterations increases, the test error decreases faster.

The uniform method reaches an error of 15.2% after 24,000 iterations (40 seconds). The Active Selection reaches 15.05% after 10,000 iterations (90 seconds) because it is able to select which examples to process and therefore discard outliers. The AST is in between: It selects more informative samples than the uniform but also invest a lot of computation to update the tree ($U = 10$).

After 10,000 iterations, the test error for active methods drastically increases (not reported by Bordes et al. 2005). This is because the Active Selection has discarded support vectors in the early steps that it would not have discarded if the last samples were processed earlier. In fact, if we run another epoch (not shown here), the test error will decrease again and reach the accuracy of the uniform baseline.

6.5 Discussion

The Active Sampling Tree structure proposed in this chapter is designed to emulate the Active Selection for LASVM, that is to select samples close to the boundary. Samples close to the boundary are indeed more informative than farther samples, including the outliers that are misclassified.

The sampling strategy with the proper weight correctly selects informative samples, and to account for the changing distribution and to avoid needing all the weights, the samples can be organised beforehand to reflect the regularity of the data.

On the Census Income data set, the result of the AST are limited: although AST selects better samples than the uniform approach (left plot of figure 6.8), this does not translate into a smaller number of support vectors relatively to the accuracy (right plot of figure 6.8).

However, as we have shown on a large image data set augmented with artificial distortions, AST proves to be competitive with the Active selection, and always reaches a better accuracy faster than the uniform approach.

7 Conclusion

Throughout this dissertation, we have investigated how to concentrate the computation of learning algorithms on the most informative samples, whether it is hard negative samples to train object detectors through a bootstrapping approach, or samples with a high loss or small classifier responses for classification.

In practice, a learning algorithm has never access to all the possible samples that could exist, but rather to a subset of them which constitute the training set. Our contributions have focused on how to efficiently choose a subset of the training samples without actually looking at all of them. In particular, when the training set is organised in a consistent manner, it is easier to make a decision on subgroups of images/samples, either to invest more computation, or on the contrary to discard them.

As we have shown in chapter 4, the adaptation of the Monte Carlo tree search is able to identify promising subgroups of images rich in hard negatives samples by properly handling the exploration-exploitation trade-off, but could easily be ported to design a web crawler to gather images. A potential criterion to take into account could be hardware constraints, related to the speed of memory, disks, and network accesses, with a view to automatically adjusting the sampling to web sites with a good throughput of informative images, for a fixed amount of time or other meaningful budget.

The tree-based structure and the sampling policies that we proposed in chapters 5 and 6 concentrate the selection of the online process on informative samples during training. However, while high gradient samples or high loss samples are informative for boosting and neural networks, low response samples are the ones informative for support vector machines. A possible future step would be to harmonise both approaches.

Our line of research has been original in the sense that most of the research carried out nowadays tend to rely on powerful hardware and massive parallelisation, and surprisingly, little research is done to design methods to concentrate more cleverly the computation and the search on the most important samples. Although it is simpler to handle data in a uniform or linear manner, we argue that investing a part of the research in this direction can be significant to improve existing techniques.

Appendices

A Sample Distillation

Contents

This chapter presents a way, called "distillation" to recursively select informative samples to reduce the size of an initial training set, using a criterion that ensures the maximisation of the information content of the selected subset. This chapter is based on the following publication:

Olivier Canévet, Leonidas Lefakis, and François Fleuret. Sample distillation for object detection and image classification. In *Proceedings of the Asian Conference on Machine Learning (ACML)*, pages 64–79, 2014

A.1 Introduction

Image classification and object detection have reached an acceptable level of performance thanks to the use of machine learning techniques with large-scale training sets. In both cases, large amounts of data points can be produced, either through artificial distortions of positive samples, or through the bootstrapping of negative samples, the objective being to enrich the available population, either in the neighbourhood of positive examples, or at the interface between the positive and the negative population, as characterised by a predictor trained on a limited data set.

In such cases, the crux of the problem is the training itself, based on these large amounts of data. Most of the learning algorithms have a computation cost at least linear with the number of samples, and are difficult to parallelise.

Interestingly, while most training sets are redundant, very few methods explicitly try to leverage that redundancy to reduce the computational cost. Some well known techniques such as the stochastic gradient descent are explicitly justified through the redundancy of samples, but no method exists that processes the data in the same spirit as a feature-selection procedure does with a feature space: by explicitly reducing the cardinality of the space, while keeping the informative content as high as possible.

We propose in this chapter to exploit a machine learning technique called “reservoir learning” by Lefakis and Fleuret (2013), which has been developed precisely to select jointly informative subsets of samples. We adapt it to a large-scale context by applying it in a recursive manner, allowing to reduce the overall computation cost by several orders of magnitude, to control its memory footprint, and to parallelise it on a multi-core architecture.

We apply this “distillation” procedure to boosting in two different contexts: pedestrian detection in natural images, where it allows to improve the set of bootstrapped negative samples, and character recognition, where distillation is used to get a better set of synthetic distortions.

A.2 Related work

Though bootstrapping is widely used in computer vision (Shotton et al., 2005), there has been very little prior work on how to efficiently mine hard negative examples. Typically once a classifier has been built, it is used to process the set of negative samples and a margin based approach is used to select which of these samples are to be added to the training set (Felzenszwalb et al., 2010c; Li et al., 2013). Other approaches are possible, as for example using non-maxima suppression (Neubeck and Van Gool, 2006; Comaniciu et al., 2002); non-maxima suppression is also widely used to handle overlapping detections but despite its prevalence there seems to be no prior experimentation with this method for hard negative sample mining. We present such results in our experiments as a baseline for our proposed method.

A different approach to representing the richness of the background class is to attempt to model it using a (normal) distribution (Osadchy et al., 2012; Hariharan et al., 2012). Negative samples can then be generated using a fitted model. Such approaches however tend to be quite slow and ignore the fact that natural images tend to have long-tailed distributions as shown by Ruderman and Bialek (1993).

A notable exception to this scarcity is the recent work by Henriques et al. (2013), that specifically addresses hard negative mining, or to be exact how to avoid it. The authors present a formulation for a specific family of classifiers that allows them to address the translations of samples in the negative sample space by modeling these translations as circular shifts. Though attaining very good results, their method is specific to translations while the method presented here handles the entire negative sample space. Furthermore, their approach cannot be used in connection with the boosting family of predictors.

The other method commonly employed in vision applications, is that of augmenting the positive sample space by random distortions of its population, sometimes referred to as jittering. This technique is used to artificially augment the number of positive samples which is oftentimes limited due to the cost of labelling. Distortions techniques used can be mirror versions of the originals, affine transformations (translations, scaling) like in the work of LeCun et al. (1998) and even elastic distortions in the work of Ciresan et al. (2012). Distortions are usually generated at random and no care is taken to choose informative ones.

In the context of boosting (Freund and Schapire, 1997), there have been many approaches proposed for data set sub-sampling (Bradley and Schapire, 2007; Domingo and Watanabe, 2000). (Lefakis and Fleuret, 2013) have proposed a novel algorithm for addressing this issue, which aims at selecting the most informative samples, in a joint manner, amongst a large number of samples. Their work focuses on trade-offs between online and offline learning, here however we aim at extending this work so as to tackle the sample set selection problems mentioned above.

A.3 Sample Distillation

We first present a brief overview of boosting and the Greedy Edge Expectation Maximisation (GEEM) algorithm proposed by Lefakis and Fleuret (2013) and subsequently present and analyse the proposed distillation method. We leave the specifics of its application to the experimental part of the paper.

A.3.1 Boosting and GEEM

Boosting algorithms greedily build an ensemble classifier H

$$H(x) = \sum_{t=1}^{T} a_t h_t(x) \tag{A.1}$$

by adding what is referred to as a weak learner $h_t(x) \in \{-1, 1\}$ at each iteration t among a pool of weak learners $\mathscr{H}$ so as to minimise an empirical loss

$$h_t = \underset{h \in \mathscr{H}}{\operatorname{argmin}} L_{\mathscr{T}}(H_{t-1} + h_t) \tag{A.2}$$

with $\mathscr{T}$ being the training data.

As mentioned, a problem that often arises when handling large data sets is that the size of $\mathscr{T}$ is such that it does not all fit in memory making the optimisation problem A.2 unwieldy or even intractable.

Typically in such cases the training set $\mathscr{T}$ is sub-sampled at each iteration t to yield a subset $\mathscr{T}^t \subset \mathscr{T}$ which is then used to acquire an estimate of the empirical loss.

The GEEM algorithm of Lefakis and Fleuret (2013) addresses the issue of optimally sampling $\mathscr{T}^t$ under certain assumptions. Specifically the authors assume that the weak learner responses on the samples follow a multivariate Gaussian distribution, they then use a greedy backward selection procedure to decide which samples to keep in $\mathscr{T}^t$, so that the expected empirical loss on $\mathscr{T}$, when using $\mathscr{T}^t$ to solve (A.2), is minimal.

In other words, given a signed (boosting) weight for each sample, the GEEM procedure selects a subset of samples such that, if a predictor has a large weighted response over the selected samples, it has a large expected weighted response over the selected samples *and* the discarded ones. The GEEM selection process accounts for correlation between predictor responses, hence allowing to discard samples whose response vectors are redundant.

A.3.2 Distillation

While GEEM has many desirable properties, and despite the efficient implementation proposed in Lefakis and Fleuret (2013), it remains extremely costly, with the cost being prohibitive for large amounts of data, exactly in the regime that is of interest to modern applications, due to its cubic dependency on the number of samples N. We propose to drastically reduce this cost, while maintaining the statistical accuracy, by processing the data recursively using a tree structure.

Given the full set of N samples $\mathscr{T}^*$, we first partition it into K disjoint subsets[1] $T_1^0, \ldots, T_K^0$ such that

$$\forall 1 \leqslant k \leqslant K, \quad |T_k| = \frac{N}{K}. \tag{A.3}$$

We then apply the GEEM procedure recursively, each time merging two subsets T_{2r-1}^d and T_{2r}^d

[1]Here the superscript denotes the tree level of a node in the tree (and by extension the subset it contains) and the subscript denotes the numbering of the node at its level.

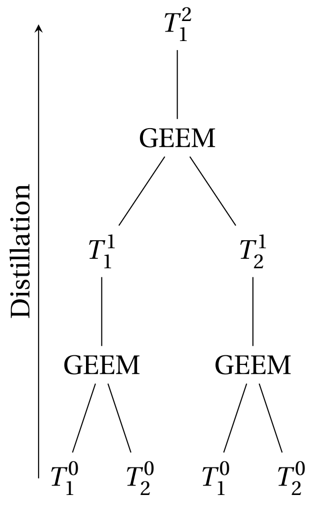


Figure A.1 – The distillation process consists in starting from a partition $T_1^0, \ldots, T_K^0$ (with $K = 4$ here for simplicity) of the full set of samples $\mathscr{T}$, then recursively concatenating subsets by pairs, and extracting one subset of fixed size $\frac{N}{K}$ from each resulting set using the GEEM procedure.

and extracting one T_r^{d+1} of half the cardinality:

$$T_r^{d+1} \subset T_{2r-1}^d \cup T_{2r}^d, \text{ s.t. } |T_r^{d+1}| = \frac{N}{K}, \tag{A.4}$$

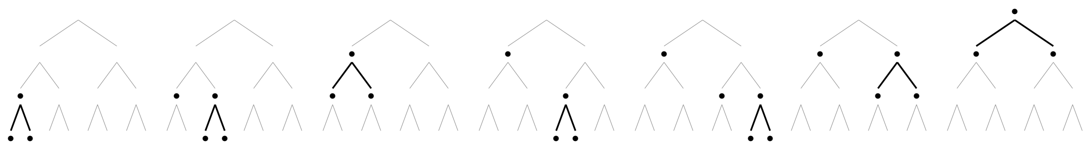
until only one subset of samples T_1^D remains.

A.3.3 Complexity and trade-offs

Lefakis and Fleuret (2013) analyse the complexity of GEEM and point out the limitations on the applicability of the algorithm due to its cubic complexity. By using the tree structure presented in the previous subsection, this cubic complexity becomes less of a restraining factor.

Whereas in reservoir boosting the complexity of the method is $O(N^3)$, N being the number of samples, in distillation the cost is $O(N^3/K^2)$ where K is the number of tree leaves. For a balanced binary tree with K leaves there are $O(K)$ nodes in the tree, each containing $O(N/K)$ samples. Thus sampling between two nodes incurs a cost of $O(N^3/K^3)$ and this process must be repeated $O(K)$ times, leading to the $O(N^3/K^2)$ complexity which translates to a K^2 complexity improvement.

Moreover the tree structure allows a significant degree of parallelisation. The sampling in a specific node of the tree can obviously be performed independently with the remaining nodes at the same level of the tree. Thus, depending on the number of CPU cores available, one can choose a trade-off between computation time and memory load. In contrast, in the case of reservoir boosting, the GEEM process cannot be similarly parallelised.


(a) Distillation on a single computation core

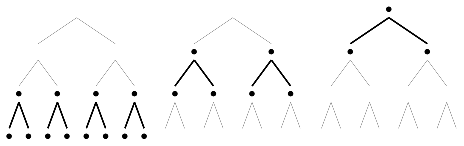

(b) Distillation on multiple computation cores

Figure A.2 – Comparison of the computation cost and memory footprint of the single-core distillation (top) and the multi-core one (bottom). Black dots picture batches of samples actually stored in memory, and bold edges the individual GEEM operations. If K is the total number of sample batches we have initially, that is the number of leaves in the distillation tree–and we have a single computation core, the memory footprint is at most $O(\log_2 K)$ (see section A.3.3) and the computation cost $O(K)$. If we have K computation cores, the memory footprint is at most $O(K)$ and the computation $O(\log_2 K)$.

Figure A.2 shows the two extreme cases of computation-memory trade-off in the distillation process. In the first case, there is only one CPU core available, and the main concern is the memory load. A naive implementation would first load all samples in the leaves and proceed up the tree level by level. Depending on the size of N this procedure might be prohibitive from a memory perspective.

The tree however can be traversed in a much more memory-efficient manner, as depicted in figure A.2(a) by always fully processing any sub-tree before starting to process a leaf belonging to another one.

Let $M(K)$ be the memory footprint of the distillation of K leaves. Obviously $M(1) = 1$, since there is nothing to do and we must have the samples of that leaf in memory. If we have to distil $2L$ leaves, we can first distil the first L, with a memory usage of at most $M(L)$, store the result for a memory usage of 1, and process the second half, for a memory usage of $M(L)$. Hence $M(2L) = 1 + M(L)$, which leads to $M(K) \sim O(\log_2 K)$

Computation-wise, since we have to run sequentially through all the nodes, the computation time is $O(K)$

In the case of K cores, shown in figure A.2(b), we can achieve a computation of $O(\log_2 K)$ by assigning each node in a level to a different core and proceeding upward in a breadth-first manner, *i.e.* first we process all nodes at one level before moving up to the next. Given that a balanced binary tree with K leaves has a height of $\log_d K$, the algorithm will terminate in time $O(\log_2 K)$. As pointed out above, loading the entire data at the leaves of the tree has a $O(K)$ memory footprint.

Thus according to the resources at hand, the distillation's tree structure allows us to choose the optimal trade-off between the two resource types.

A.4 Pedestrian detection

We first apply the distillation procedure to bootstrapping negative samples for object detection.

A.4.1 Data set

We use the INRIA Person data set (Dalal and Triggs, 2005) which is commonly used to train and evaluate pedestrian detectors. For training, the data consists of 2,416 64×128 cropped colour images of pedestrians and a set of 1,218 pedestrian-free scenes. To evaluate the performances, the test set consists of 1,132 pedestrians and 453 scenes. The scenes are used to extract the negative and the false positive samples in the first training and in the bootstrapping phase. We use histograms of oriented gradients (HOG, Dalal and Triggs 2005; Felzenszwalb et al. 2010c) as a descriptor for the images with the parameter values specified in Dalal and Triggs (2005).

A.4.2 Bootstrapping

Let $\mathscr{T}_+$ be the set of all available positive samples in $\mathscr{T}$, and $\mathscr{T}_-^*$ the set of all available negative samples. In the case of detection, there are usually a few thousands samples in the former, and millions or more in the latter.

In bootstrapping, we first train an initial two-class predictor f^0 with $\mathscr{T}_+$ and a subset $\mathscr{T}_-^0 \subset \mathscr{T}_-^*$, obtained by sampling uniformly in $\mathscr{T}_-^*$ without replacement.

The objective of both our distillation method and all the baselines is to use f^0 to select a subset $\mathscr{T}_-^1 \subset \mathscr{T}_-^*$ such that we can build a second predictor f^1 with $\mathscr{T}_+$ and $\mathscr{T}_-^1$, more accurate than f^0.

In all the techniques we consider, the number of selected negative samples N_- is a constant fraction $\zeta \in [0, 1]$ of the total number of false positives:

$$N_- = |\mathscr{T}_-^1| = \left\lceil \zeta \left| \left\{ x \in \mathscr{T}_-^*, \text{ s.t. } f^0(x) \geqslant 0 \right\} \right| \right\rceil \tag{A.5}$$

where $\lceil\rceil$ is the ceiling function (*i.e.* smallest integer greater than).

A.4.3 Performance evaluation

We compare our approach of selecting negative samples among the false positives with the following methods. The algorithms described below are applied per-scene.

Uniform sampling consists in sampling N_- samples uniformly without replacement in the subset of false positives, that is, according to the distribution

$$\forall x \in \mathscr{T}_-^*,\ p_x = \frac{1}{Z}\mathbf{1}_{\{f^0(x)\geqslant 0\}}, \tag{A.6}$$

where Z is a normalisation constant.

Max-sampling consists in selecting in $\mathscr{T}_-^*$ the N_- samples with highest f^0 responses, that is the ones which are the most "incorrectly classified":

$$\forall x \in \mathscr{T}_-^1, x' \in \mathscr{T}_-^* \setminus \mathscr{T}_-^1,\ f(x) \geqslant f(x'). \tag{A.7}$$

Weighted sampling consists of sampling N_- samples in $\mathscr{T}_-^*$, without replacement, according to the exponential loss, that is with the distribution

$$\forall x \in \mathscr{T}_-^*,\ p_x = \frac{1}{Z}\mathbf{1}_{\{f^0(x)\geqslant 0\}}e^{f^0(x)}, \tag{A.8}$$

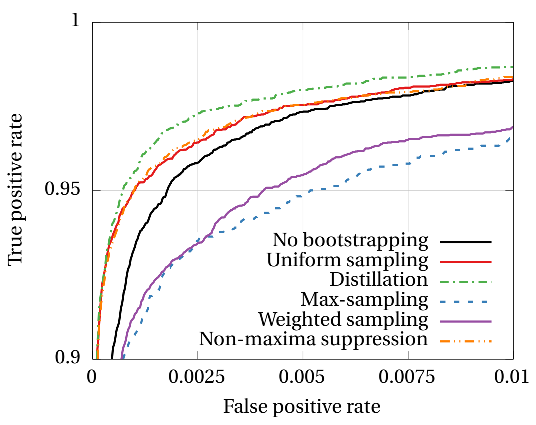
where Z is a normalisation constant. This method is somewhat similar to the max-sampling but also allows to pick samples of small individual weights which have a large cumulated weight together.

Non-maxima suppression consists in selecting the samples with the highest f^0 responses and then discarding an area around the selected one to prevent from sampling in its neighbourhood again. This method is used in object detection to remove multiple detections and in this case will tend to pick samples equally spaced in the image.

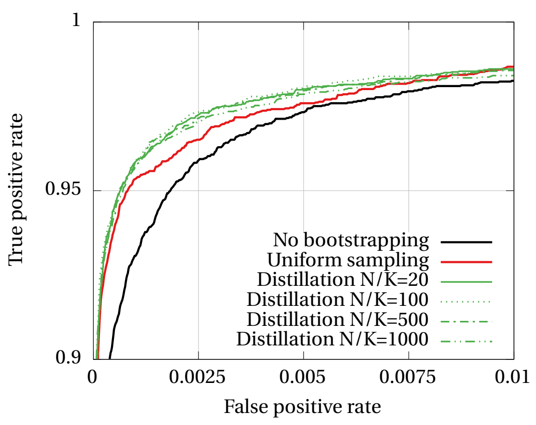
Figure A.3a shows the receiver operating characteristic (ROC) curves of the trained predictors.

The initial predictor is trained with $P = 2,416$ positive images and $N = 10,000$ negative ones. We then retain 3% of the false positive per scene (or at least 10), which lead on average to $H \simeq 18,000$ hard negative samples. Both predictors are trained using AdaBoost for 1,000 rounds on the HOG features. The curves are averaged over 5 runs.

The predictor trained using max-sampling performs worse than the initial one. The weighted sampling method also performs worse than the initial predictor for the same reason but does better than max-sampling because it is still able to sample among false positive with smaller weights. The uniform sampling and the non-maxima suppression perform comparatively better than the initial predictor. These methods are able to populate the training set with difficult samples so that the boosting procedure focuses on the interface between positive and negative classes. Finally, the distillation method does even better than the uniform sampling. Distillation is able to discard the false positive samples that do not bring anymore information to the predictor given the ones it has already picked. We can thus say that it has built a richer training set.



(a) Performance of the classifier



(b) Influence of N/K

Figure A.3 – Bootstrapping images to enrich the training set with difficult examples. On figure A.3a, the max-sampling and weighted sampling perform worse than no bootstrapping because they tends to put emphasis on redundant samples. The uniform sampling and the non-maxima suppression perform the same and our distillation process outperforms all other sampling methods. Figure A.3b shows the influence of the size of the subsets in the distillation. The greater the size, the closer to the true GEEM the process is.

Figure A.3b depicts the influence of K in the distillation process. As all the scenes do not contain the same number of false positives, fixing K across all scenes would produce subsets of different sizes. Instead, we fix N/K (size of the subset) to be constant. As $N/K \to N$, the distillation gets closer to the true GEEM. The results show here that surprisingly, the size does not really matter. However, we have noticed that to enforce good diversity in the final distilled set, scenes should be processed separately.

A.5 Character recognition

We now apply the GEEM procedure to enrich a training set with informative distortions for image classification.

A.5.1 Data set

We use the MNIST data set (LeCun et al., 1998). The train set consists of 60,000 28×28 gray scale images of digits from 0 to 9 and the test set contains 10,000 images. We use Haar like wavelets (Viola and Jones, 2004) as features without pre-processing the images, as in the work of Kégl and Busa-Fekete (2009) where AdaBoostMH achieves 1.02% test error.

Method	Test error (%)
No distortions	1.2 (+0.0361)
+1 random distortion per image	0.962 (± 0.0616)
+1 distilled distortion per image	0.9075 (+0.0585)

Table A.1 – Classification error on the full MNIST data set. The GEEM procedure is once again able to choose better sample in comparison to random. We can make the same observation as for the reduced sets: adding too much GEEM samples does not do better than random (here more than 3)

A.5.2 Sub-sampling distortions

In the case of jittering we seek to augment a relatively small set of positive samples $\mathscr{T}_+$ by randomly distorting an image and adding the resulting image to the pool of positive samples.

Suppose we create such a set of artificial positive samples $\mathscr{T}_+^{'}$, we can either create a set of small cardinality $|\mathscr{T}_+^{'}| = N_+$ or we can create multiple distortions resulting in a set of large cardinality and then downsample to N_+, using GEEM.

Again we first train an initial two-class predictor f^0 with $\mathscr{T}_+$ and $\mathscr{T}_-$ and then create a set $\mathscr{T}_+^{'}$ of cardinality $N^{'} \gg N_+$. We then use f_0 in connection with the GEEM process to sub-sample $\mathscr{T}_+^{'}$ down to $\mathscr{T}_+^1$.

We then use $\mathscr{T}_+^1$ together with $\mathscr{T}_+$ and $\mathscr{T}_-$ to build the predictor f^1.

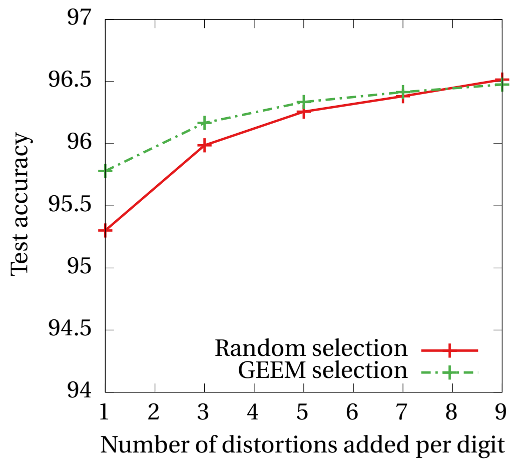
A.5.3 Performance evaluation

The images are distorted using the combination of translations of 1 pixel and rotation of angle $\alpha \in [-10, 10]$. Using higher shifts or angles produces images which are too far from the true distribution of images. Test images are not distorted at test time.

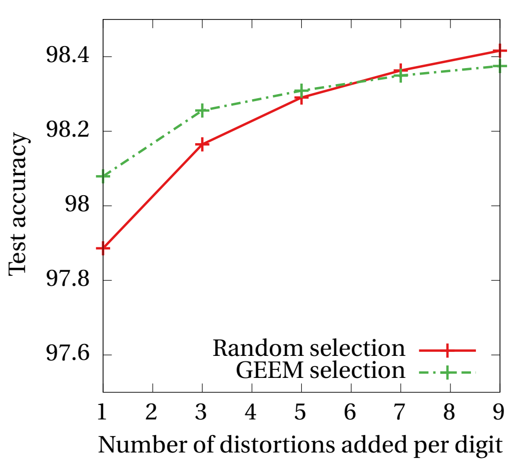
As GEEM requires a initial classifier we train one on the non-distorted set. In our experiments the training set is augmented by n distortions per digit, that is for a set of S images and n distortions per image, we train the second classifier with $(n+1)S$ images.

We first analyse the performances of the GEEM procedure on a reduced training set of 1,000 and 5,000 images. Results for full scale experiments are shown in table A.1.

Figure A.4 shows that the GEEM procedure is able to build a more informative training set than the random. The test accuracy is higher. The reason lies in how the boosting loss is reduced with this GEEM. Figure A.5 show the loss of validation samples in the early iterations of the boosting algorithm when training with the new augmented training set. The validation samples were chosen among the discarded distortions for the GEEM.



(a) With 1,000 images from MNIST



(b) With 1,000 images from MNIST

Figure A.4 – Analysis of the training set augmented with distortions. The graph shows the test accuracy as a function of the number of distortions added per digit (5 distortions means that the second classifier was trained with $N + 5N$ images). The initial test accuracy for 1,000 (resp. 5,000) is 94.47% (resp. 97.37%). First we see that adding distorted images helps. Second, we see that the GEEM procedure is able to pick "better" samples when few are added (from 1 to 7) but that random does better after that. See figure A.5 for an explanation from the loss point of view.

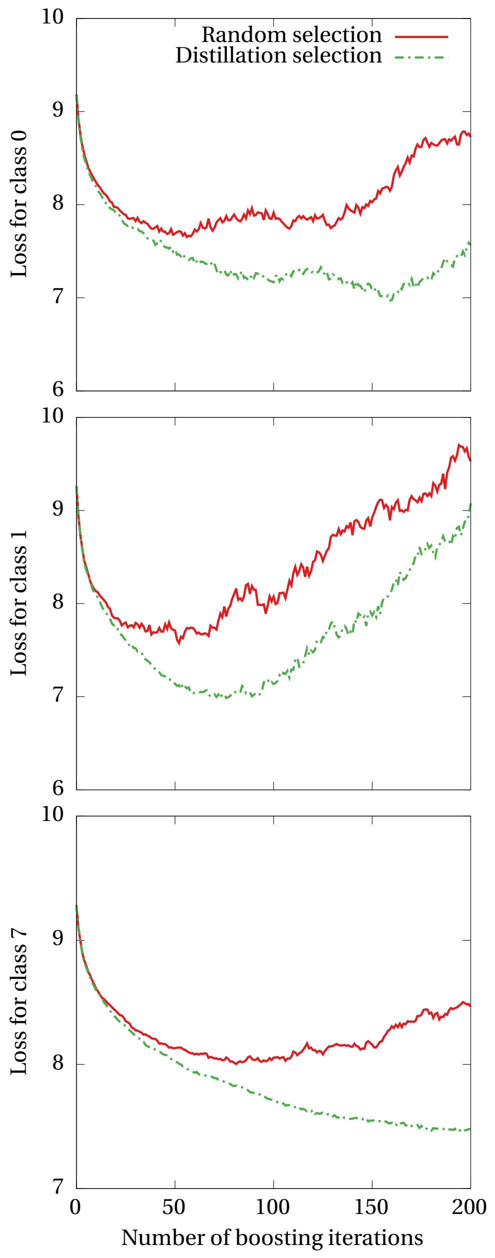
A.5.4 Analysis of the GEEM behavior

To understand a bit further why GEEM is able to build a more informative training set, we analyse its behaviour in specific settings.

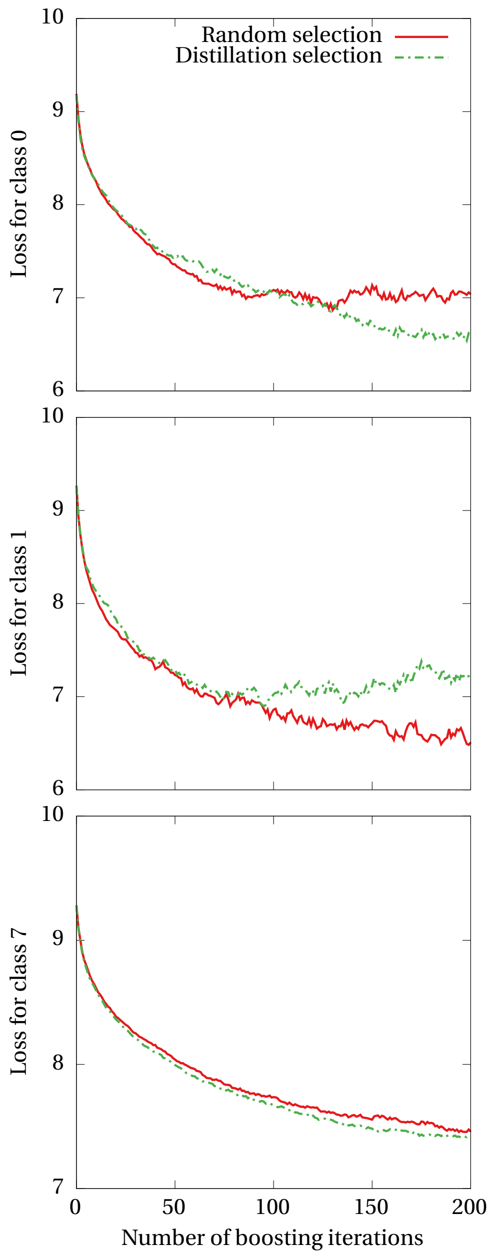
How does GEEM handle similar images?

To analyse how GEEM behave with similar images, we select 5 images from the same class and duplicate the first one 124 times (in this set, images from index 1 to 124 are the same, and images 125 to 128 are all different). We run the GEEM procedure to discard images down to 6 and note which indexes are kept. We run this experiment 1,000 times, each time selecting a new class and 5 images of the class and count how many times each index was kept by the process.

The histogram in Figure A.6 shows how many times the images were selected on average over the 1,000 runs. We see that the last 4 indexes (*i.e.* the images which were unique in the set) tend to be selected more often than any of the duplicated one. Since we select 6 images out of 5 unique ones, the duplicated one is at least selected twice at each round. The GEEM procedure is therefore able to build a diverse set of samples that can be used for training another classifier.



(a) Adding 3 deformations per digit



(b) Adding 7 deformations per digit

Figure A.5 – Analysis of the loss on validation samples for random and GEEM training samples. GEEM aims at discarding samples whose boosting response can already be predicted with selected samples. Here we analyse the boosting loss of some validation samples. These samples were discarded by the GEEM procedure. We see that their loss is even more reduced than training with a set of random distortions. Figure A.4 show that adding more than 7 GEEM distortions does not help. This is what we can see on figure A.5b, class 1 where the loss is not as low for GEEM.

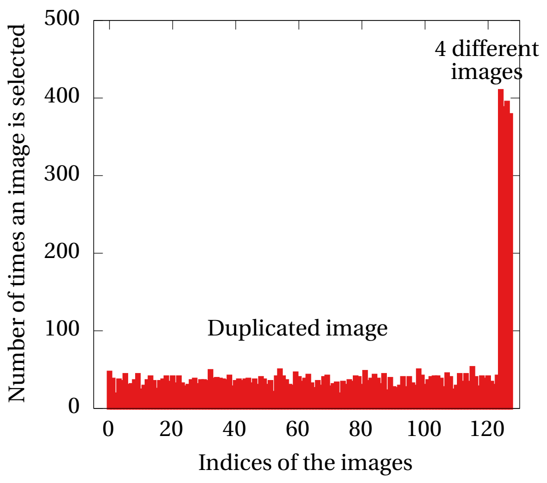


Figure A.6 – Analysis of the selection of images in a set containing duplicates. In this experiment, we build a set of 128 MNIST images of the same class from 5 different images and duplicate the first one 124 times. The set is thus 124 times the same image (indices from 1 to 124) alongside 4 different images (indices from 125 to 128). We run the distillation process to select 6 images out of the 128 with buckets of size 16. We run this experiment 1,000 times selecting randomly a class each time and count how many times the images where chosen. We see that the 4 unique images tend to be selected more often than the duplicates which shows that the GEEM procedure manages to build a diverse set of images.

How does GEEM handle distortions?

We analyse how GEEM behaves in comparison to randomness. First, we choose one image at random, generate one random distortion and compute the ℓ_2 norm between them. Second, we pick one image at random, generate 100 random distortions (one of which is the original) and use GEEM to discard all of them but one and compute the the ℓ_2 norm between the original and the remaining distortion. We repeat this experiment 5,000 times.

The histogram of figure A.7 shows the distribution of the ℓ_2 norm between the original and the distortion for both methods. We see that in comparison to the random set-up, the GEEM procedure will tend to choose a distortion different from the original which once again shows that GEEM can build a diverse set of samples given the initial classifier.

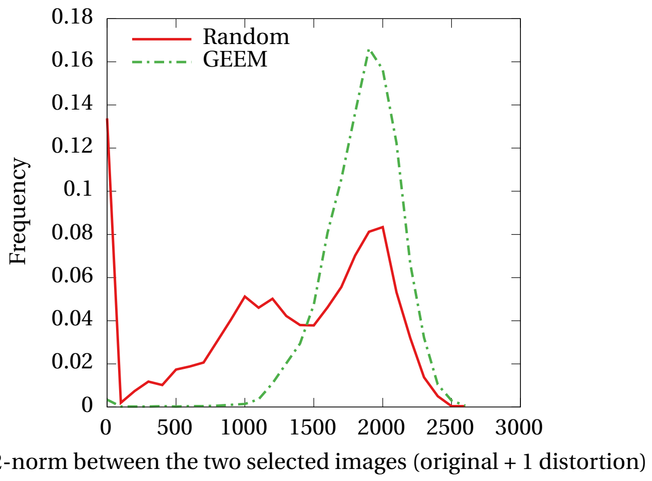


Figure A.7 – Analysis of the difference between selecting random distortions and distilled distortions to augment the training set. In this experiment, we compute the ℓ_2 norm between an original image and a distortion either selected randomly or through distilling it from a set of 100 distortions. We run the experiment 5,000 times and plot a histogram of the distances between the original image and the chosen distortion. The figure shows that the distillation process tends to select images which are different from the original leading to diversity. The peak observed for the random part is due to the fact that we do not interpolate the grey levels of the distorted image. When the rotation angle is too small, the resulting image is the same as the original. This does not occur for the distillation process because it manages not to select duplicates (see experiment of figure A.6).

A.6 Conclusion

We presented a novel algorithm that extends the GEEM algorithm to make it applicable to large-scale data. The proposed extension not only allows the algorithm to scale-up but also allows for a computation/memory trade-off in its application due its ability to be processed in a parallel manner.

Through experimental results we show the relevancy of the method for computer vision by extending the state-of-the-art in two important applications, namely bootstrapping for object detection and data set augmentation through distortions.

We furthermore analysed the method not only from a theoretical perspective, *i.e.* its complexity, but also experimentally showing that the method not only performs well but in fact has a number of desirable properties that actually appear in practice.

In future work we plan to extend the distillation algorithm to work with a greater family of predictors, in particular with support vector machines. We are especially interested in the applicability of such approaches to the family of deformable part models.

Another line of current research is a more thorough theoretical analysis of jointly informative subset selection and in particular its relation to batch active learning.

B Finding “super” examples

Contents

We present an analysis of the image classifier proposed by Coates and Ng (2011). Given a trained image classifier, we are interested in finding “super” images, that is images on which the given classifier has a high response. A similar work has been conducted by Nguyen et al. (2015) on convolutional neural networks in order to generate images that the network classify with more than 99.99% confidence, although not reminiscent of the class for a human eye at all.

B.1 Motivation

At testing time, an image is given the label of the classifier which has the highest response among them. More formerly, the predicted label $\hat{c}$ of image x is

$$\hat{c} = \underset{1\leqslant c\leqslant C}{\operatorname{argmax}} f_c(x) \tag{B.1}$$

Note that values $f_c(x)$ can be negative for all c. In the 1-vs-all scheme, it means that all the classifiers f_c predicted the negative class, but we take the “less” negative. In our implementation, all scores f_c lay in the range $[-2;2]$, so a well classified horse with a strong response will have a score around 2.

We would like to find an image x_c^* with a high score for a given class c. This is an interesting problem because it answers the question: “Does a very high response image for a given class look like the object for which the classifier is trained?” It is equivalent to solve the following optimisation problem:

$$x_c^* = \underset{x}{\operatorname{argmax}} f_c(x), \tag{B.2}$$

which can be solved by using gradient descent. Starting from a initial image, the iterative algorithm updates the new image x alongside its gradient:

$$\text{at step } t, \quad x^t = x^{t-1} - \epsilon \nabla f_c(x^{t-1}) \tag{B.3}$$

where ϵ controls the speed of the displacement.

B.2 Implementation

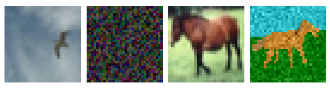
Given the fact that $f_c(x)$ encapsulates the patch extraction, the whitening process and the pooling step, computing its gradient is not practical. As a result, we used finite differences as an approximation: the gradient of $f_c(x^t)$ for $x^t \in \mathbb{R}^N$ is

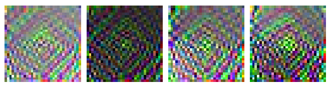
$$\nabla f_c(x^t) = [\partial_1 f_c(x^t), \dots, \partial_N f_c(x^t)]^T, \tag{B.4}$$

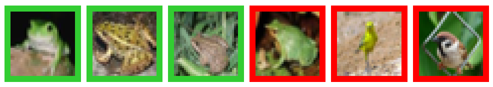
and we approximate it such that:

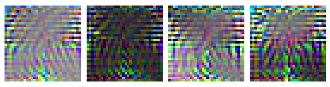
$$\forall\, 1 \leqslant n \leqslant N, \quad \partial_n f_c(x^t) \simeq \frac{f_c(x_1^t, \dots, x_n^t + d, \dots, x_N^t) - f_c(x_1^t, \dots, x_n^t - d, \dots, x_N^t)}{2d}. \tag{B.5}$$

x_i^t denotes the pixels of image x^t. d represents the local neighbourhood on which the partial derivative is approximated. Due to the whitening transformation and the pooling step, we choose $d = 10$ to avoid having a null component. And as we are dealing with integers in $[\![0,255]\!]$,


(a) Initial images


(b) Optimal image for class frog


(c) Top well/bad classified images for class frog


(d) Optimal image for class horse

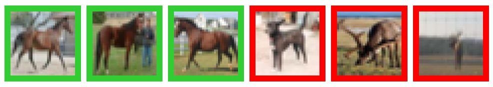

(e) Top well/bad classified images for class horse

Figure B.1 – Images in B.1a are the initial images used for gradient descent. The images below are the ones to which the algorithms have converged after 300 iterations of gradient descent, when the target class was frog (images B.1b) and horse (images B.1d). Pictures B.1c and B.1e are the images of the test set of CIFAR with a high response in the two considered classes. A green frame indicates that the image is correctly classified as a frog/horse, and the red frame indicated that the image is misclassified. We note that even though the bird of picture B.1c has a positive score in the class bird, it has a bigger score in class frog, possibly due to the perch.

we choose ϵ such that at least one pixel is changed at each iteration:

$$\epsilon = \frac{10}{||\nabla f_c(x^t)||_1}. \tag{B.6}$$

B.3 Results

Figure B.1 shows the results of the gradient descent after 300 iterations for each of the four initial images that we considered. It is interesting to see that apart from brightness (which seems to be determined from the initial image), all procedures converge to the same image for a given target class. Moreover, a high response image in a class is made of specific frequencies which are reminiscent of the objects of the class. For instance, the bottom part of the optimal image for class horse is made of vertical lines. As a result, an image made of such patterns in this area will tend to have a high score for this class. This is shown in figure B.1e where the animal legs are clearly visible in this part of the images. Note that the bird in figure B.1c has a positive score for class bird, but a greater one in class frog, which we can reasonably put down to the presence of the square perch.

Figure B.2 shows the scores of the different classifiers as the number of iterations of the gradient descent grows. The classification score of the image x^t is increasing for the class we are targeting and remains small in the others.

This experiment is interesting to carry out because it shows that the classification pipeline

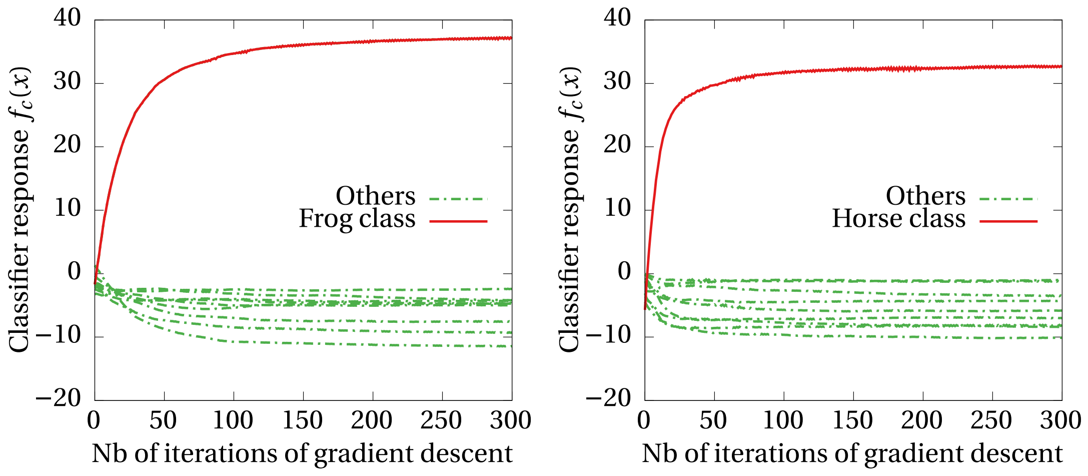


Figure B.2 – Classifier responses as the number of iterations grows. In both graphs, the red curve is the response of the classifier which we want to maximise (frog or horse) and the green curves are the responses of the other classifiers. As expected, the score of the target class (no matter what is the initial image) increases with the number of iterations, while the others responses remain small and negative. We also note that this high response can be around 30, while at test time, a high response of a real image is usually between 1 and 2.

tends to classify the objects according to their poses. Most of the frogs look like the second example and most of the horses look like the three green-framed ones.

Bibliography

Shipra Agrawal and Navin Goyal. Analysis of thompson sampling for the multi-armed bandit problem. *arXiv preprint arXiv:1111.1797*, 2011.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

K. Bache and M Lichman. Covertype dataset. `https://archive.ics.uci.edu/ml/datasets/Covertype`.

Antoine Bordes, Seyda Ertekin, Jason Weston, and Léon Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579–1619, September 2005.

Lubomir Bourdev and Jonathan Brandt. Robust object detection via soft cascade. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 236–243. IEEE, 2005.

Amine Bourki, Guillaume Chaslot, Matthieu Coulm, Vincent Danjean, Hassen Doghmen, Jean-Baptiste Hoock, Thomas Hérault, Arpad Rimmel, Fabien Teytaud, Olivier Teytaud, et al. Scalability and parallelization of monte-carlo tree search. In *Computers and Games*, pages 48–58. Springer, 2011.

Joseph K. Bradley and Robert E. Schapire. Filterboost: Regression and classification on large datasets. In *NIPS*, 2007.

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, Simon Colton, et al. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012.

Olivier Canévet and François Fleuret. Efficient sample mining for object detection. In *Proceedings of the Asian Conference on Machine Learning (ACML)*, pages 48–63, 2014.

Olivier Canévet and François Fleuret. Large scale hard sample mining with Monte Carlo Tree Search. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Olivier Canévet, Leonidas Lefakis, and François Fleuret. Sample distillation for object detection and image classification. In *Proceedings of the Asian Conference on Machine Learning (ACML)*, pages 64–79, 2014.

Olivier Canévet, Cijo Jose, and François Fleuret. Importance sampling tree for large-scale empirical expectation. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.

Alex J. Champandard. Monte-carlo tree search in total war: Rome ii's campaign ai, 2014. URL `http://aigamedev.com/open/coverage/mcts-rome-ii/`.

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. In *Advances in neural information processing systems*, pages 2249–2257, 2011.

GMJB Chaslot, Jahn-Takeshi Saito, Bruno Bouzy, JWHM Uiterwijk, and H Jaap Van Den Herik. Monte-carlo strategies for computer go. In *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium*, pages 83–91. Citeseer, 2006.

Guillaume Chaslot, Christophe Fiter, Jean-Baptiste Hoock, Arpad Rimmel, and Olivier Teytaud. Adding expert knowledge and exploration in monte-carlo tree search. In *Advances in Computer Games*, pages 1–13. Springer, 2010.

Dan Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

Adam Coates and Andrew Y Ng. The importance of encoding versus training with sparse coding and vector quantization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 921–928, 2011.

Ronan Collobert, Samy Bengio, and Yoshua Bengio. A parallel mixture of svms for very large scale problems. *Neural computation*, 14(5):1105–1114, 2002.

Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

Dorin Comaniciu, Peter Meer, and Senior Member. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24: 603–619, 2002.

Pierre-Arnaud Coquelin and Rémi Munos. Bandit algorithms for tree search. 2007.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3): 273–297, 1995.

Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and games*, pages 72–83. Springer, 2007.

Gabriella Csurka, Christopher Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. In *Workshop on statistical learning in computer vision, ECCV*, volume 1, page 22, 2004.

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

Thomas Dean, Mark Ruzon, Mark Segal, Jonathon Shlens, Sudheendra Vijayanarasimhan, and Jay Yagnik. Fast, accurate detection of 100,000 object classes on a single machine. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1814–1821, 2013.

Piotr Dollár, Z. Tu, Pietro Perona, and Serge Belongie. Integral channel features. In *BMVC*, 2009.

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

Piotr Dollar, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: An evaluation of the state of the art. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(4):743–761, 2012.

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

Carlos Domingo and Osamu Watanabe. Madaboost: A modification of adaboost. In *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory*, COLT '00, pages 180–189, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-703-X. URL `http://dl.acm.org/citation.cfm?id=648299.755176`.

Charles Dubout and François Fleuret. Exact acceleration of linear object detectors. In *Computer Vision–ECCV 2012*, pages 301–311. Springer, 2012.

Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, 2015.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

Li Fei-Fei and Pietro Perona. A bayesian hierarchical model for learning natural scene categories. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 524–531. IEEE, 2005.

Pedro F Felzenszwalb, Ross B Girshick, and David McAllester. Cascade object detection with deformable part models. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 2241–2248. IEEE, 2010a.

Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1627–1645, 2010b.

P.F. Felzenszwalb, R.B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1627–1645, Sept 2010c. ISSN 0162-8828. doi: 10.1109/TPAMI.2009.167.

Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems. *J. Mach. Learn. Res*, 15(1): 3133–3181, 2014.

F. Fleuret and D. Geman. Fast face detection with precise pose estimation. In *Proceedings of the IEEE International Conference on Pattern Recognition (ICPR)*, volume 1, pages 235–238, 2002.

F. Fleuret and D. Geman. Stationary features and cat detection. *JMLR*, 9:2549–2578, 2008.

Francois Fleuret and Donald Geman. Coarse-to-fine face detection. *International Journal of computer vision*, 41(1-2):85–107, 2001.

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

Kristen Grauman and Trevor Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1458–1465. IEEE, 2005.

Giovanni Gualdi, Andrea Prati, and Rita Cucchiara. Multi-stage sampling with boosting cascades for pedestrian detection in images and videos. In *Computer Vision–ECCV 2010*, pages 196–209. Springer, 2010.

Bharath Hariharan, Jitendra Malik, and Deva Ramanan. Discriminative decorrelation for clustering and classification. In *Proceedings of the 12th European Conference on Computer*

Vision - Volume Part IV, ECCV'12, pages 459–472, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-33764-2. doi: 10.1007/978-3-642-33765-9_33. URL `http://dx.doi.org/10.1007/978-3-642-33765-9_33`.

Joao F Henriques, Joao Carreira, Rui Caseiro, and Jorge Batista. Beyond hard negative mining: Efficient detector learning via block-circulant decomposition. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 2760–2767. IEEE, 2013.

Cho-Jui Hsieh, Si Si, and Inderjit S Dhillon. Fast prediction for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, pages 3689–3697, 2014.

David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591, 1959.

Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE, 2009.

Cijo Jose, Prasoon Goyal, Parv Aggrwal, and Manik Varma. Local deep kernel learning for efficient non-linear svm prediction. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 486–494, 2013.

Z. Kalal, J. Matas, and K. Mikolajczyk. Weighted sampling for large-scale boosting. In *BMVC*, 2008.

Balázs Kégl and Róbert Busa-Fekete. Boosting products of base classifiers. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 497–504. ACM, 2009.

Dan Klein and Christopher D Manning. A parsing: fast exact viterbi parse selection. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 40–47. Association for Computational Linguistics, 2003.

Donald E Knuth and Ronald W Moore. An analysis of alpha-beta pruning. *Artificial intelligence*, 6(4):293–326, 1976.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293. Springer, 2006.

Martin Koestinger, Paul Wohlhart, Peter M. Roth, and Horst Bischof. Annotated facial landmarks in the wild: A large-scale, real-world database for facial landmark localization. In *First IEEE International Workshop on Benchmarking Facial Image Analysis Technologies*, 2011.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.

Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2169–2178. IEEE, 2006.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

L. Lefakis and F. Fleuret. Reservoir boosting : Between online and offline ensemble learning. In *Proceedings of the international conference on Neural Information Processing Systems (NIPS)*, 2013.

X. Li, C. G. M. Snoek, M. Worring, D. C. Koelma, and A. W. M. Smeulders. Bootstrapping visual categorization with relevant negatives. *IEEE Transactions on Multimedia*, 15(4):933–945, June 2013.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014*, pages 740–755. Springer, 2014.

Gaëlle Loosli, Stéphane Canu, and Léon Bottou. Training invariant support vector machines using selective sampling. *Large scale kernel machines*, pages 301–320, 2007.

David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

Markus Mathias, Rodrigo Benenson, Marco Pedersoli, and Luc Van Gool. Face detection without bells and whistles. In *Computer Vision–ECCV 2014*, pages 720–735. Springer, 2014.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

Rémi Munos. From bandits to monte-carlo tree search: The optimistic principle applied to optimization and planning. 2014.

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

D. Needell, N. Srebro, and R. Ward. Stochastic Gradient Descent, Weighted Sampling, and the Randomized Kaczmarz algorithm. *ArXiv e-prints*, oct 2013.

Alexander Neubeck and Luc Van Gool. Efficient non-maximum suppression. In *Proceedings of the 18th International Conference on Pattern Recognition - Volume 03*, ICPR '06, pages 850–855, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2521-0. doi: 10.1109/ICPR.2006.479. URL `http://dx.doi.org/10.1109/ICPR.2006.479`.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 427–436. IEEE, 2015.

Margarita Osadchy, Daniel Keren, and Bella Fadida-Specktor. Hybrid classifiers for object classification with a rich background. In *Proceedings of the 12th European Conference on Computer Vision - Volume Part V*, ECCV'12, pages 284–297, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-33714-7. doi: 10.1007/978-3-642-33715-4_21. URL `http://dx.doi.org/10.1007/978-3-642-33715-4_21`.

Marco Pedersoli, Andrea Vedaldi, and Jordi Gonzalez. A coarse-to-fine approach for fast deformable object detection. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1353–1360. IEEE, 2011.

John C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, Advances in Kernel Methods - Support Vector Learning, 1998.

Herbert Robbins. Some aspects of the sequential design of experiments. *Bull. Amer. Math. Soc.*, 58(5):527–535, 09 1952. URL `http://projecteuclid.org/euclid.bams/1183517370`.

Daniel L. Ruderman and William Bialek. Statistics of natural images: Scaling in the woods. In *NIPS*, pages 551–558, 1993.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Andrew Saxe, Pang W Koh, Zhenghao Chen, Maneesh Bhand, Bipin Suresh, and Andrew Y Ng. On random weights and unsupervised feature learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1089–1096, 2011.

Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

Jamie Shotton, Andrew Blake, and Roberto Cipolla. Contour-based learning for object detection. In *Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1 - Volume 01*, ICCV '05, pages 503–510, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2334-X-01. doi: 10.1109/ICCV.2005.63. URL `http://dx.doi.org/10.1109/ICCV.2005.63`.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587): 484–489, 2016.

Patrice Simard, Bernard Victorri, Yann LeCun, and John Denker. Tangent prop-a formalism for specifying selected invariances in an adaptive network. In *Advances in neural information processing systems*, pages 895–903, 1992.

Patrice Y Simard, Yann A LeCun, John S Denker, and Bernard Victorri. Transformation invariance in pattern recognition—tangent distance and tangent propagation. In *Neural networks: tricks of the trade*, pages 239–274. Springer, 1998.

I. Steinwart and A. Christmann. *Support vector machines*. Springer Science & Business Media, 2008.

William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, pages 285–294, 1933.

Antonio Torralba, Rob Fergus, and William T Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(11):1958–1970, 2008.

Vladimir Vapnik and Alexander Lerner. Pattern recognition using generalized portrait method. *Automation and remote control*, 24:774–780, 1963.

Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

Stefan Walk, Nikodem Majer, Konrad Schindler, and Bernt Schiele. New features and insights for pedestrian detection. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 1030–1037. IEEE, 2010.

A.J. Walker. New fast method for generating discrete random numbers with arbitrary frequency distributions. *Electronics Letters*, 10(8):127–128, April 1974. ISSN 0013-5194. doi: 10.1049/el:19740097.

Markus Weber. Caltech background dataset. `http://www.robots.ox.ac.uk/~vgg/data/background/background.tar`.

Junjie Yan, Zhen Lei, Longyin Wen, and Stan Li. The fastest deformable part model for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2497–2504, 2014.

Bin Yang, Junjie Yan, Zhen Lei, and Stan Z Li. Aggregate channel features for multi-view face detection. In *Biometrics (IJCB), 2014 IEEE International Joint Conference on*, pages 1–8. IEEE, 2014.

P. Zhao and T. Zhang. Stochastic optimization with importance sampling. *arXiv preprint arXiv:1401.2753*, 2014.

Olivier Canévet

olivier.canevet@idiap.ch

www.idiap.ch/~ocanevet

EDUCATION

Idiap Research Institute – École Polytechnique Fédérale de Lausanne *Oct. 2012 – Present*
PhD candidate in Computer Vision and Machine Learning *Martigny, Switzerland*
Subject: *Object detection with Active Sample Harvesting*
Advisor: François Fleuret

TELECOM Bretagne *Sep. 2007 – Sep. 2012*
Graduate Engineering School *Brest, France*
The school trains non-specialized engineers in image & information processing, computer science, networks and electromagnetism.

Master of Science Degree SISEA *Sep. 2011 – Mar. 2012*
Image processing

Vienna University of Technology (TU Wien) *Feb. 2009 – Jun. 2009*
ERASMUS semester: electronics, networks and computer science. *Vienna, Austria*

B.S. in mathematics *Sep. 2007 – Jun. 2008*

Pierre Corneille High School *Sep. 2005 – Jun. 2007*
Advanced mathematics and physics classes in preparation for the competitive examination granting admission to the French "Grandes Ecoles" (graduate engineering schools)

French Scientific Baccalaureate (High School Diploma) with highest honors *2005*

EXPERIENCE

CERN (European Organization for Nuclear Research) Apr. 2012 – Sep. 2012
Master internship *Geneva, Switzerland*

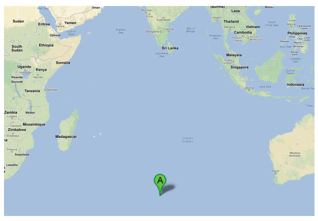
- I worked on a ranking method in Invenio which aims at aggregating several ranking scores coming from different methods into a new one. In addition to query-related scores, the goal was to take into account the results of previous searches to rank higher the most relevant and famous documents in the database.
- python, invenio, SQL

CNES (French Space Agency) Apr. 2011 – Sep. 2011
Final project intership *Toulouse, France*

- I built a supervised procedure to improve the segmentation of a satellite image to extract objects of a scene by detecting over-segmentation. The developments were made with the Orfeo Toolbox (C++, ITK).
- Presented the work on a poster during the Pleiade Days (January 17-18th, Toulouse)
- I presented the work during a conference on remote sensing (IGARSS 2012)
- C++, OTB

French Polar Institut Sep. 2009 – Mar. 2011
Technical voluntary service *Amsterdam island, Indian ocean*

- I was in charge of the seismological and magnetic observatories of the EOST Strasbourg on a isolated scientific base (for one year)
- I did daily manual measurements of the magnetic field using a theodolite,
- I make sure of the continuous recording of the data,
- I processed and sent data, available for all global observatories.



SCHOOL PROJECTS

Sep. – Dec. 2008	Development of a software program for a library to manage subscriptions, 2 persons book/DVD loans and addition of new works (40 hours) Skills: Java, Eclipse, UML
Feb. – Jun. 2008	Development of an on-board camera system to be placed in the gondola of an 2008 indoor airship (Zeppelin) Microchip programming in C language, design of an integrated circuit with Orcad, development of a computer interface with LabVIEW to guide the airship wirelessly.
2006	Study of the astronomical clock of Ploërmel built by Brother Bernardin in 1855. I exposed a method (continued fractions, using Maple) that he may have used to determine the number of teeth of the cogwheels. No one knows how he did it because his calculations were lost in 1903.

SKILLS

Programming	C++, C, Python, Matlab, lua
Computer science	Linux, bash, git/mercurial, OTB/ITK/VTK
Image	Image Segmentation, dimension reduction, Object Detection, Image Enhancement, Image Classification
Machine Learning	SVM, Boosting

PUBLICATIONS

O. Canévet and F. Fleuret. *Large Scale Hard Sample Mining with Monte Carlo Tree Search*. CVPR, 2016.

O. Canévet, C. Jose, and F. Fleuret. *Importance Sampling Tree for Large-scale Empirical Expectation*. ICML, 2016.

O. Canévet and F. Fleuret. *Efficient Sample Mining for Object Detection*. ACML, 2014.

O. Canévet, L. Lefakis, and F. Fleuret. *Sample Distillation for Object Detection and Image Classification*. ACML, 2014.

J. Michel, M. Grizonnet, and O. Canévet *Supervised re-segmentation for very high-resolution satellite images*. IGARSS, 2012.

O. Canévet. *L'horloge astronomique de Ploërmel*. Quadrature no. 72, 2009.