This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the version available on IEEE Xplore.

Large Scale Hard Sample Mining with Monte Carlo Tree Search

Olivier Canévet[1,2] and François Fleuret[1]
[1]Idiap Research Institut, Switzerland
[2]École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
{olivier.canevet,francois.fleuret}@idiap.ch

Abstract

We investigate an efficient strategy to collect false positives from very large training sets in the context of object detection. Our approach scales up the standard bootstrapping procedure by using a hierarchical decomposition of an image collection which reflects the statistical regularity of the detector's responses.

Based on that decomposition, our procedure uses a Monte Carlo Tree Search to prioritize the sampling toward sub-families of images which have been observed to be rich in false positives, while maintaining a fraction of the sampling toward unexplored sub-families of images. The resulting procedure increases substantially the proportion of false positive samples among the visited ones compared to a naive uniform sampling.

We apply experimentally this new procedure to face detection with a collection of ∼100,000 background images and to pedestrian detection with ∼32,000 images. We show that for two standard detectors, the proposed strategy cuts the number of images to visit by half to obtain the same amount of false positives and the same final performance.

1. Introduction

Learning techniques for object detection require very large sets of negative examples, which are usually used through a bootstrapping procedure. The training process constructs a sequence of predictors of increasing performance, each trained from a fixed set of positive samples and a collection of so called "hard" negative samples that fool the previous predictor.

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. From a computational perspective, bootstrapping decouples the selection of the interesting (negative) samples from their use for training the model. The latter usually has a cost linear with the number of selected samples, which is far less than the number of samples in the full set. However, the selection process from the full set requires one evaluation of the predictor per candidate sample, and remains linear with the total number of samples.

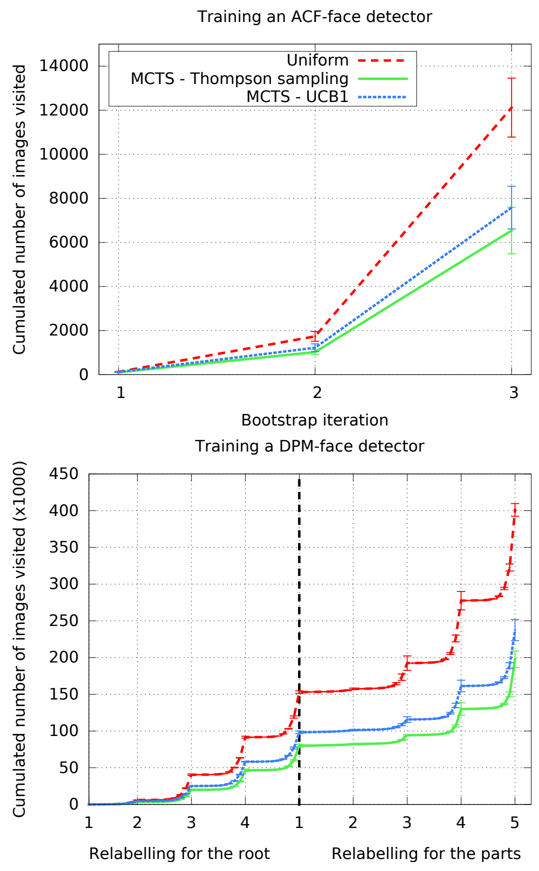


Figure 1: Number of images visited vs. bootstrapping iterations for a the ACF-face detector (top) and a DPM-face detector (bottom). Our methods using a Monte-Carlo tree search (blue and green) focus on difficult images and visit roughly half the number of images the traditional approach needs (red).

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

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 sub-sets, image) it should make a decision about investing or not more computation in the corresponding samples.

We propose to formalize 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 dataset related images in the top level (dataset, origin, etc., see figure 2 for an example). If the structure is given (temporal structure, keywords, etc.) then no pre-processing is needed. Given such a tree and an existing predictor, each leaf is labeled 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 maximize 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: 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-arm bandit to each node of the tree, and uses them to sample paths down the tree based on the current estimates of rewards, or in our case, of proportion of hard samples in the sub-trees. While MCTS is traditionally used to characterize 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.

2. Related works

2.1. Object detection and bootstrapping

Object detection aims at predicting the position and the scale of all the instances of an object class in an input image. Most detectors use a binary classifier which discriminates the object from the background, and evaluates it at all positions and scales in the image. Multiple detections are removed with a non-maxima suppression post-processing.

The binary classifier is trained with a population of positive samples corresponding to location and scales in images where the object is visible, and a population of negative samples uniformly taken in (parts of) images where the object of interest is not visible. Then a bootstrapping approach [30] is used to improve the classifier by assembling better negative sample sets: The training set is augmented by a collection of misclassified samples and used to train a new classifier. This procedure emphasizes difficult samples that lie at the boundary between the two classes and can be repeated multiple times. A hard sample can be defined as being on the wrong side of the boundary [32] or in the margin of the classifier [16, 13].

The number of bootstrapping steps varies depending on the complexity or the nature of the classifier and on the number of hard samples that are added: Dalal and Triggs [10] perform only one step of bootstrapping but add all the false positives that are found until it no longer fits in memory. The pedestrian detector of Dollar *et al.* [13] is trained in a soft cascade fashion with three rounds of bootstrapping, each time adding 5,000 new samples. The deformable part based model (DPM) [16] is trained with a maximum of ten rounds for each re-labeling of the positive samples, with a stopping criterion based on the variation of the objective function. Finally when a detector is trained in a cascade fashion, the bootstrapping procedure is applied at each level. Henriques *et al.* [18] showed that in the particular case of linear classifiers, the Gram matrix of translated samples can efficiently be computed with non overlapping windows which allows to train with the fully translated set of samples without any bootstrapping step.

Most of the works in object detection have focused on proposing new features to improve the performances of the final detector such as Haar-wavelets [32], histograms of oriented gradients (HOG) [10], optical-flow based features and self-similarity [33] or aggregated channel features (ACF) [13]. Besides, other works have concentrated on speeding up the detection at test time. Cascades [32, 3] or coarse-to-fine approaches [17, 28] allow to reject many windows in the early stages and concentrate most of the computation on promising parts of the image. In the case of the DPM, the convolutions can be efficiently computed in the Fourier domain [15] or hashed into tables for fast access [11, 29].

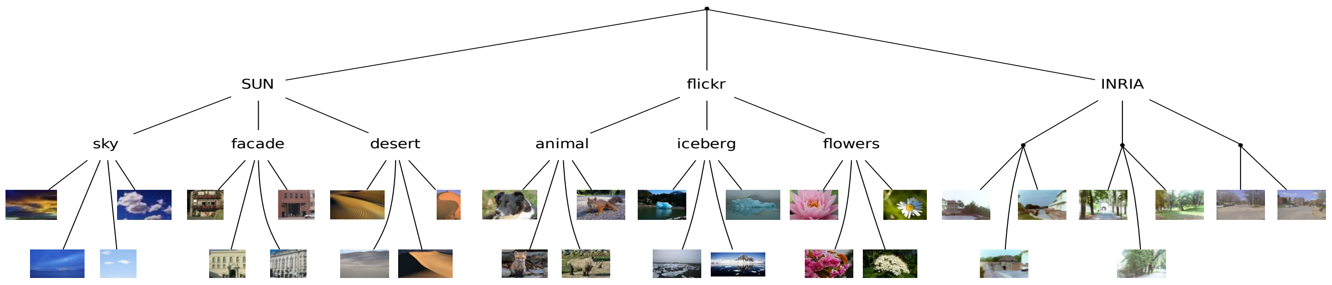


Figure 2: Example of a structured image database. The structure can be explicit such as in the SUN dataset (facade, sky, etc.) or implicit like in the INRIA Person dataset, where images can be grouped either based on their names or after a pre-clustering of the images. Videos have a temporal structure.

Nevertheless, little work has been done to efficiently build the set of hard samples during the bootstrapping step. For instance, the works by Canévet *et al*. [5] and Kalal *et al*. [19] sample respectively in the image plan, and in sample sets, without prior structure over the databases nor explicit concern for the exploration/exploitation trade-off. The structure of the data-set is never exploited in order to find the hard samples faster.

2.2. Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search is a method to find the optimal solution in a given and potentially huge search space [4]. MCTS balances between analyzing 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 $\alpha\beta$ [21] or A^* [20] have shown to be efficient against humans for the game of chess or checker because it is quite easy to evaluate the outcome of the game given the current state. But for games such as Go or Backgammon, computers have long been unable to defeat non-professional players until MCTS appeared. By doing randomized 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 [9, 22, 7] and computers are now able to defeat human on small boards for the game of Go.

Another example of successful use of MCTS is the optimization of a “black-box” function [27, 8] 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. The space is split in a hierarchical manner, and MCTS determines in which subspace the function should be evaluated next. As the process goes

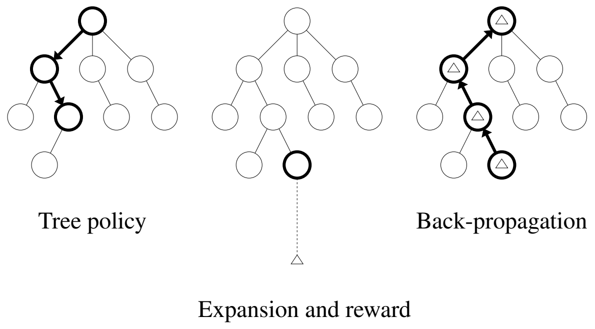


Figure 3: One run of the traditional MCTS [4]

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. We now describe more precisely the basic run of MCTS in 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 winning configurations. Using this tree, the sampling procedure of MCTS (see Figure 3) 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 [2] 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 back-propagated 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/time budget is exhausted, at which point the next move is made by selecting the best child of the root which is the one with

Algorithm 1 UCB1 [2]

```
$\forall\, k,\ \bar{X}_k \leftarrow 0, n_k \leftarrow 0$
**for** $t = 1$ to $T$ **do**
    Select arm $k_t = \mathrm{argmax}_k\ \bar{X}_k + \sqrt{\frac{2 \ln t}{n_k}}$
    Observe reward $X_t$ from arm $k_t$
    Update $\bar{X}_{k_t}$ and $n_{k_t}$
**end for**
```

the maximum proportion of wins.

MCTS and bandit algorithms have nice theoretical properties. In particular, one such property is 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 § 3.2, we propose in this paper to formulate the problem of mining hard samples in a MCTS way. We associate to each leaf an image, 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.

3. Method

For clarity, we first recall below the basics of multi-arm bandits and Monte Carlo Tree Search. Then, in § 3.3, we present how we adapt such strategies to bootstrapping.

3.1. Multi-armed bandit (MAB)

As explained in section 2.2, the selection of the next child to visit is formulated as a MAB problem, that we now describe using the analogy of a gambler in a casino. Given a slot machine with K arms, at each iteration $t \in [\![1, T]\!]$, the player selects one arm and plays it. This generates a reward following an unknown distribution described by θ_k with support in $[0, 1]$, and of unknown expectation μ_k. The arm with the largest expectation is called optimal and is the one that the player would play all the time, had he knew it was optimal. The goal of the player is to maximize his cumulative payoff, or equivalently, to minimize his cumulative regret, that is the loss due to not playing the optimal machine all the time. Lay and Robbins [24] proved that the regret grows at least logarithmically with the number of plays. Solving the MAB problem consists in finding a policy to select the next arm to pull given past observations and to achieve a logarithmic regret.

Many algorithms have been proposed to select the best arm at a given iteration [2]. Although the usual policy used for the MAB in MCTS is UCB1, we will also present Thompson sampling because the reward we aim at modeling – namely the proportion of false positives – is strongly biased toward very small values which is inconsistent with the standard assumptions justifying the use of UCB1.

Algorithm 2 Thompson Sampling [6]

```
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 $k_t = \mathrm{argmax}_k\, \mathbb{E}[X_k \mid \theta_k]$
5:     Observe reward $X_t$ from arm $k_t$
6:     $D \leftarrow D \cup (k_t, X_t)$
7: **end for**
```

3.1.1 Upper Confident Bound (UCB1)

The UCB1 [2] selects the arm maximizing

$$\bar{X}_k + \sqrt{\frac{2 \ln t}{n_k}},$$

where $\bar{X}_k$ is the average reward of arm k (estimated from the previous plays), n_k the number of times arm k has been played and t the total number of plays done so far. The first term is the *exploitation* term and is larger for arms with more rewards. The second term is the *exploration* term and tends to be larger for less frequently pulled arms. As the exploration term is a decreasing function of the time, the beginning of the process is dominated by the exploration term while the end of the process is driven by the exploitation one. Asymptotically, only the best arms are pulled.

3.1.2 Thompson sampling for MAB (TS)

Thompson Sampling [31] was introduced to address the exploration/exploitation trade-off in a purely Bayesian manner. It was applied recently for the MAB problem in [6], and then proved to achieve a logarithmic regret [1]. The idea of TS 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, that is choosing the arm maximizing $\mathbb{E}[X_k|\theta_k]$, where θ_k is drawn from the posterior at each iteration.

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

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. Algorithms 1 and 2 summarize the selection of the next arm for both presented strategies.

3.2. Monte Carlo Tree Search (MCTS)

In MCTS, the traversal of the tree is performed from top to bottom. Starting from the root node, one iteration of MCTS consists in associating a MAB on the children of the current node, selecting the best child (that is the one maximizing the score of MAB), and going further down with the new selected node. When the policy reaches a leaf, a reward is drawn from it.

If the MAB policy is UCB1 (§ 3.1.1), the child selected is the one maximizing

$$\bar{X}_i + \sqrt{\frac{2 \ln p}{n_i}},$$

where p (resp. n_i) is the number of times the current node (resp. child i) has been visited.

If the MAB policy is based on TS (§ 3.1.2), the next child is chosen by sampling according to its posterior probability, based on the passed traversals of the current node.

The reward is then back-propagated and the statistics of the nodes between the leaf and the root are updated (see figure 3). The number of visits are incremented; for UCB1, the mean of the nodes are recomputed given the outcome of the simulation (win or loss) and for TS, the posterior is updated for future draws.

The MCTS policy will asymptotically visit the best branches that lead to more wins leaving unexpanded non promising parts of the search space. The MCTS framework seems therefore well suited for the task of mining hard samples in a large structured collection of images.

We next present how MCTS is ported to bootstrapping. We define the rewards obtained at the leaves (*i.e.* in the images) after detecting false positives and how the tree is updated to avoid going back to the same images twice.

3.3. MCTS Bootstrapping

3.3.1 Image dataset structure

As said in the introduction, image datasets inherently have a hierarchical structure by the way images were collected, and our procedure builds upon this structure (see figure 2).

In the top level, the children of the root corresponds to a specific image dataset, such as Pascal, INRIA, or any image directory available on one's hard disk.

Further down the tree, nodes would correspond to subparts of each datasets, such as the year for Pascal (2007, 2008, etc.) or the name of the semantic object contained in the sub-directory for SUN (desert, abbey, etc.). Finally, at the bottom of the tree, leaves are individual images.

When an image dataset comes with no explicit structure (such as Microsoft Coco [25] or INRIA Person [10]), a pre-clustering can be applied build sub-groups of visually similar images, which correspond in practice to coarse semantic categories of similar structural complexity (see § 4.2.2). This is what is depicted by figure 2 below the "INRIA" node, where images taken from the same place (city, forest, etc.) have a common ancestor.

3.3.2 Procedure

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

After training the initial detector with the collection of positive samples and a collection of negative samples uniformly taken in the dataset, 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 dataset 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 MAB selects the dataset from which the image will be chosen. Then a second MAB selects from which sub-part of this dataset, 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 back-propagated 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 dataset, that is on sub-groups 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.

3.3.3 Scores

As described in § 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 $[0, 1]$ [2].

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

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

This "win" score corresponds to the vanilla setup of the MAB where the reward is either 0 or 1 ("did the player win or not?"). The back-propagation rule to update the score of node p with the scores of its children $\mathcal{C}(p)$ is $\text{win}_p = \sum_{c \in \mathcal{C}(p)} \text{win}_c$.

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 size does not have the same (computational) cost. A natural score would be $d = h/S$, 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 $\sim$100,000 times on a 640×480 image so finding 10 hard samples leads to a reward of 0.0001. The exploration term dominates the small exploitation score and there is no exploitation.

We thus normalize this score so that it ranges in $[0, 1]$ and finally, we define

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

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 will be 0.5, and when an image is rich in false positives (*i.e.* h/S much larger than Z), the score will be close to 1. The score is back-propagated with $\tilde{d}_p = \sum_{c \in \mathcal{C}(p)} \tilde{d}_c$.

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 3 different policies for the MAB: two of them for UCB1 and one for TS.

3.3.4 MCTS-strategies

MCTS-UCB1-win (win) The first strategy is based on UCB1 (§ 3.1.1) with the "win" score. At a given node p, the next node to select among its children $\mathcal{C}(p)$ is the one maximizing

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

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 *normalized* density of hard samples and the next child

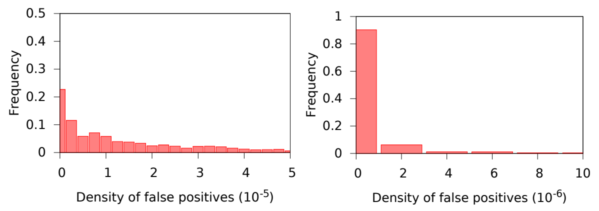


Figure 4: Distribution of false positives (Left: bootstrap 2, right: bootstrap 3)

to be selected is the node maximizing

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

where $\tilde{d}_c$ is the *normalized* density of false positives as described in 3.3.3.

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 (§ 3.1.2) as described in [6], a model of the distribution of the observations is required as well as a prior on its parameter. As previously stated, TS does not require the rewards to lie in $[0, 1]$ so we here directly use the *true* density.

Some simple experiments (see figure 4) 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 $p(D|\lambda) = \lambda e^{-\lambda x}$ (with notations of algorithm 2, $\theta = \lambda$). 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 TS 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 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. Experiments

We present the results of our experiments to train a face detector and a pedestrian detector with a large dataset of images. We show that our MCTS-based bootstrapping approach is able to leverage the tree structure of the dataset to efficiently find hard samples. We will make our code publicly available at the time of publication to allow the reproduction of the experiments.

4.1. Detectors

The ACF detector [13] belongs to the state-of-the-art detectors for pedestrian detection [34] and face detection [26, 36]. It consists of a series of channels which are combined with boosted small depth trees. We used in part the implementation provided by the original author [12].

The DPM [16] also reaches state-of-the-art performance for face detection [26, 35] and we used the Fourier-based implementation of [15].

4.2. Image datasets with a tree structure

4.2.1 CaltechPedestrians

The pedestrian detector is trained on the CaltechPedestrians dataset [14] as described in [34, 12] by using $24,498$ positive examples and 3 rounds of bootstrapping each time adding $25,000$ hard samples. The structure of the tree is pretty straight forward because of the temporal structure of the dataset. 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.

4.2.2 Face-free images

The face detector ACF (resp. DPM) is trained with $15,000$ (resp. $7,000$) images of faces from AFLW [23] 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 [25]. 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 2). As for Coco and Pascal, there is no inherited structure. To make one, we perform a pre-clustering of these datasets: We make a 48×64 thumbnail of each image, compute its features (gradient or GIST) and recursively perform a k-mean clustering.

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 dataset.

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. This justifies the use of the number of images instead of the number of evaluation of the detector as a reference.

	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 1: Number of images visited (normalized to 640×480) for each strategy to train an ACF face detector with 3 bootstrapping steps (averaged over 10 runs). The performance on Pascal Faces and AFW is the average precision (AP). The *s indicate which strategies are also depicted in figure 1.

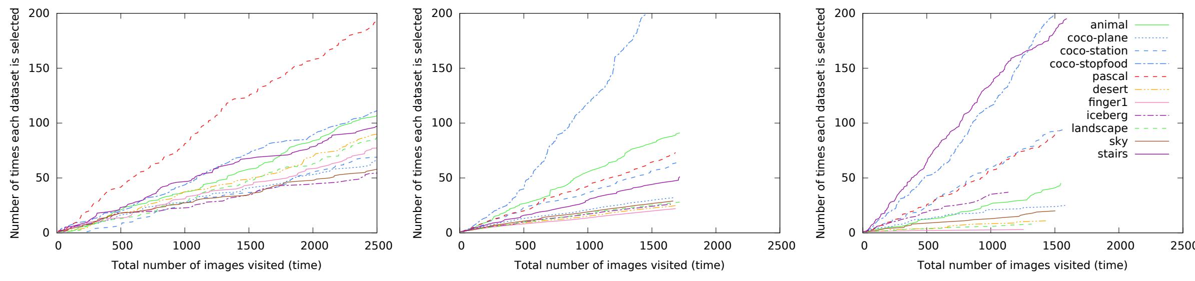
	Nb images visited			Miss rate @ fppi		
Strategy	Boot1	Boot2	Boot3	0.01	0.1	1
uniform	568	2143	13767	0.46	0.29	0.16
win	563	2132	8885	0.46	0.30	0.16
dense	536	1843	7683	0.49	0.31	0.17
ts	509	1874	8625	0.48	0.31	0.17

Table 2: Number of images visited to train an ACF pedestrian detector (averaged over 10 runs). The temporal structure of the video was kept to build the tree. The performance on the CaltechPedestrians test set is the miss rate at a given number of false positive per image (fppi).

4.3.1 Reduction of the number of visited images

The main performance measure for the proposed methods, that is the number of images to visit to collect a required number of hard samples, is presented in tables 1 and 2. Our MCTS-based methods need roughly half as many images than the traditional uniform approach, without hurting the performance of the resulting trained detector (last columns).

The top plot of figure 1 summarizes table 1 by showing the cumulated number of images required to train an ACF-face detector. Regarding DPM, we only present the cumulated number of images on the bottom plot of figure 1 for space consideration. For all re-labeling steps of the positive examples, MCTS strategies require half as many images as the uniform approach.



(a) Traditional uniform selection (b) MCTS-UCB1-dense (dense-clust) (c) MCTS-Thompson (ts-clust)

Figure 5: These plots show how often the datasets are visited by the procedure during the second bootstrapping step to collect $5,000$ hard samples with the ACF detector. The uniform method selects a dataset proportionally to the number of images in the set (to be uniform over the entire dataset) and requires more than $2,500$ images to find enough false positives (x axis cut). Our MCTS bootstrapping approaches identify the most promising datasets (Coco, Pascal, animals) while leaving 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) dataset, MCTS is also able to discard sub-branches and visit it less often.

4.3.2 Behavior of the sampling

Figure 5 shows how often the datasets are visited over time when training an ACF-face detector. The uniform approach selects images uniformly in the whole dataset, that is proportionally to the number of images in each sub-sets. The Pascal dataset is more visited because it contains $7,537$ images while the other sets have around $3,000$. Each dataset is visited linearly with time. Unsurprisingly the MCTS approaches identify the datasets Pascal, Coco, animals or stairs are being rich in hard samples and visit them more often. Datasets of sky, desert of fingerprints are quickly identified as being useless. The speed-up is therefore due to the identification of good branches of the dataset.

When MCTS methods are applied on a pre-clustered dataset (suffix “clust” and “gist” in table 1) the speed-up is even more than on non-clustered dataset. This is due to the fact that in Coco and Pascal sets, the pre-clustering put uninformative images in the same clusters such as planes over a blue sky or landscapes. MCTS thus identifies uninformative sub-branches. We did not use this pre-clustering step on CaltechPedestrians because the temporal structure is consistent in itself. MCTS on a shuffled data-set (suffix “shuf”) performs as poorly as the uniform approach.

Figure 6 shows how many times a sequence of Caltech-Pedestrians was selected as a function of its average number of hard samples. 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 speed-up.

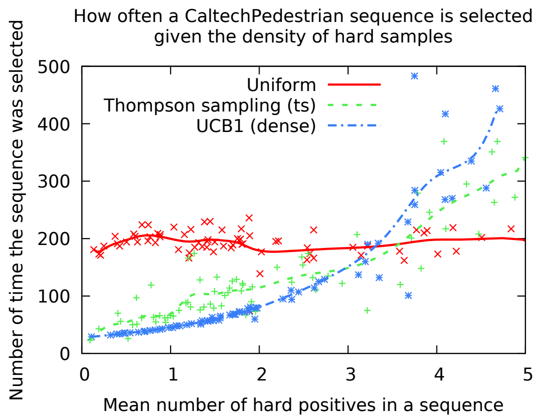


Figure 6: All sequences are equally visited by the uniform approach whereas richer sequences in hard samples are much more selected by the bandit based strategies.

5. Conclusion

We propose a novel approach to collect hard negatives from large databases of images. Instead of visiting images in a uniform and unstructured manner, we use a hierarchical structure that goes from the level of collections of databases down to the individual images, and to leverage that architecture with a bandit-base exploration strategy. This formulation ensures a proper scalability by relying on sound procedures for balancing exploration and exploitation.

Experiments on a collection of more than $100,000$ images show that this approach properly concentrates computation on “good images” and reduces the number of samples to visit to find the same amount of false positives by a factor of two for 2 types of detectors, ACF and DPM.

MCTS is well suited to concentrate properly computational resources on large databases for machine learning at large. It could be combined with active learning, or extended to other tasks where the availability of hand-provided labels is not critical, for instance large-scale unsupervised feature learning.

Acknowledgements This work was supported by the Swiss National Science Foundation (grant 200021-140912 – DASH)

References

[1] S. Agrawal and N. Goyal. Analysis of thompson sampling for the multi-armed bandit problem. *arXiv preprint arXiv:1111.1797*, 2011. 4

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002. 3, 4, 5

[3] L. Bourdev and J. 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. 2

[4] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012. 3

[5] O. Canévet and F. Fleuret. Efficient sample mining for object detection. In *Proceedings of the Asian Conference on Machine Learning (ACML)*, pages 48–63, 2014. 3

[6] O. Chapelle and L. Li. An empirical evaluation of thompson sampling. In *Advances in neural information processing systems*, pages 2249–2257, 2011. 4, 6

[7] G. Chaslot, J.-T. Saito, B. Bouzy, J. Uiterwijk, and H. J. 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. 3

[8] P.-A. Coquelin and R. Munos. Bandit algorithms for tree search. In *Uncertainty in Artificial Intelligence*, 2007. 3

[9] R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and games*, pages 72–83. Springer, 2007. 3

[10] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005. 2, 5

[11] T. Dean, M. A. Ruzon, M. Segal, J. Shlens, S. Vijayanarasimhan, and J. Yagnik. Fast, accurate detection of 100,000 object classes on a single machine. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 1814–1821. IEEE, 2013. 2

[12] P. Dollár. Piotr's Image and Video Matlab Toolbox (PMT). http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html. 7

[13] P. Dollár, R. Appel, S. Belongie, and P. Perona. Fast feature pyramids for object detection. *PAMI*, 2014. 2, 7

[14] P. Dollar, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: An evaluation of the state of the art. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(4):743–761, 2012. 7

[15] C. Dubout and F. Fleuret. Exact acceleration of linear object detectors. In *Computer Vision–ECCV 2012*, pages 301–311. Springer, 2012. 2, 7

[16] 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, 2010. 2, 7

[17] F. Fleuret and D. Geman. Coarse-to-fine face detection. *International Journal of Computer Vision (IJCV)*, 41(1/2):85–107, 2001. 2

[18] J. Henriques, J. Carreira, R. Caseiro, and J. Batista. Beyond hard negative mining: Efficient detector learning via block-circulant decomposition. In *proceedings of the IEEE International Conference on Computer Vision*, 2013. 2

[19] Z. Kalal, J. Matas, and K. Mikolajczyk. Weighted sampling for large-scale boosting. In *BMVC*, pages 1–10, 2008. 3

[20] D. Klein and C. 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. 3

[21] D. E. Knuth and R. W. Moore. An analysis of alpha-beta pruning. *Artificial intelligence*, 6(4):293–326, 1976. 3

[22] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293. Springer, 2006. 3

[23] M. Kostinger, P. Wohlhart, P. M. Roth, and H. Bischof. Annotated facial landmarks in the wild: A large-scale, real-world database for facial landmark localization. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 2144–2151. IEEE, 2011. 7

[24] T. L. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985. 4

[25] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *Computer Vision – ECCV 2014*, pages 740–755. Springer, 2014. 5, 7

[26] M. Mathias, R. Benenson, M. Pedersoli, and L. Van Gool. Face detection without bells and whistles. In *Computer Vision–ECCV 2014*, pages 720–735. Springer, 2014. 7

[27] R. Munos. From bandits to Monte-Carlo Tree Search: The optimistic principle applied to optimization and planning. *Foundations and Trends in Machine Learning*, 7(1):1–130, 2014. 3

[28] M. Pedersoli, A. Vedaldi, and J. 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. 2

[29] M. A. Sadeghi and D. Forsyth. 30hz object detection with dpm v5. In *Computer Vision–ECCV 2014*, pages 65–79. Springer, 2014. 2

[30] K.-K. Sung and T. Poggio. Example-based learning for view-based human face detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(1):39–51, 1998. 2

[31] W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933. 4

[32] P. Viola and M. Jones. Robust real-time object detection. *IJCV*, 4:34–47, 2001. 2

[33] S. Walk, N. Majer, K. Schindler, and B. Schiele. New features and insights for pedestrian detection. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 1030–1037. IEEE, 2010. 2

[34] J. H. H. Woonhyun Nam, Piotr Dollár. Local decorrelation for improved pedestrian detection. In *NIPS*, 2014. 7

[35] J. Yan, Z. Lei, L. Wen, and S. Z. Li. The fastest deformable part model for object detection. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 2497–2504. IEEE, 2014. 7

[36] B. Yang, J. Yan, Z. Lei, and S. Z. Li. Aggregate channel features for multi-view face detection. In *Biometrics (IJCB), 2014 IEEE International Joint Conference on*, pages 1–8. IEEE, 2014. 7