Interpretable Visual Models for Human Perception-Based Object Retrieval

Ahmed-Riadh Rebai, Alexis Joly, Nozha Boujemaa

Interpretable Visual Models for Human Perception-Based Object Retrieval

Ahmed REBAI
INRIA Rocquencourt
78153 Le Chesnay
France
Ahmed.Rebai@inria.fr

Alexis JOLY
INRIA Rocquencourt
78153 Le Chesnay
France
Alexis.Joly@inria.fr

Nozha BOUJEMAA
INRIA Rocquencourt
78153 Le Chesnay
France
Nozha.Boujemaa@inria.fr

ABSTRACT

Understanding the results returned by automatic visual concept detectors is often a tricky task making users uncomfortable with these technologies. In this paper we attempt to build humanly interpretable visual models, allowing the user to visually understand the underlying semantic. We therefore propose a supervised multiple instance learning algorithm that selects as few as possible discriminant local features for a given object category. The method finds its roots in the *lasso* theory where a L_1-regularization term is introduced in order to constraint the loss function, and subsequently produce sparser solutions. Efficient resolution of the lasso path is achieved through a boosting-like procedure inspired by BLasso algorithm. Quantitatively, our method achieves similar performance as current state-of-the-art, and qualitatively, it allows users to construct their own model from the original set of patches learned, thus allowing for more compound semantic queries.

Categories and Subject Descriptors

I.4.8 [**IMAGE PROCESSING AND COMPUTER VISION**]: Scene Analysis—*Object recognition*

General Terms

Algorithms

Keywords

Object retrieval, interpretability, feature selection, sparsity, human perception, interactivity.

1. INTRODUCTION

Understanding an image scene is definitely related to recognizing what it is composed of. Unfortunately, in practice, the results returned by state-of-the-art visual concept detectors are often difficult to interpret from a user point of view. The visual models produced are indeed highly depending on the used training data and might convey a different semantic than the words used to describe the originally targeted concept. This often makes users uncomfortable with these technologies since they do not get what they expected from the textual description of the trained concept. In this paper we rather attempt to build humanly interpretable visual models, allowing the user to visually understand the underlying semantic.

Our basic idea is to build *concise*, effective and visually *interpretable* models to allow for fast retrieval. Each object model consists of a concise set of visual patches representing the most discriminant image regions of a concept. Consequently, users can easily visualize the trained image regions and query the system by only choosing the visual patches that most correspond to their needs. This involves adding, subtracting and attributing weights to the visual patches displayed.

Composing a statistically and visually discriminant model provides a unique way for an interactive search. Now, the challenging question to answer is how to train the most concise set of visual patches while preserving a good retrieval effectiveness. The key to this question is provided by the image representation and the learning technique used. Using state-of-the-art bag-of-features technique is for instance not adapted because it discards the positions of the features being learned. These visual words do not provide any guaranty on whether the underlying clusters pertain to tangible parts of the objects (i.e eye, tooth, finger, etc.) or if they are just a statistical combination of some of these parts. On the other hand, using learning techniques based on such features with a SVM classifier is also not adequate for the matter since our objective is to locate a few representative image regions rather than creating an optimal separation between the positive and negative feature sets. It is worth mentioning that the method we propose is generic in the sense that it could be used with any local features or feature sets representing the content of an image region. In this work, we simply used SIFT features as the basic image primitives.

The main contribution of this paper is a novel multiple-instance learning method that allows for user interactivity when retrieving visual concepts. Applying the principle of the lasso technique to a discriminative approach for concept retrieval constitutes our second contribution. The third contribution consists in adapting the BLasso algorithm to our purpose by using weighted training data and constraining the forward steps to be only *positive* steps. As a last contribution, we propose an efficient implementation for our learning method to speed up the process of backward steps. The

ICMR '11, April 17-20, Trento, Italy

next section briefly reviews state-of-the-art feature selection methods related to our work. Section 3 explains our learning strategy in more detail. Then, we present our experiments in section 4 and set out our conclusions in section 5.

2. FEATURE SELECTION

Feature selection has always been an overwhelming task. The method we propose is based on a modified version of the BLasso algorithm. Our original idea is to train with a boosting-like procedure but with adding extra constraints to the loss function in order to produce sparser solutions. Added to highlighting interest features, sparsity is indeed preferable because it reduces the model complexity and subsequently enhances the interpretability of the produced models as well as it reduces the prediction time.

Constraining the empirical loss dates back to 1996 when Tibshirani observed that the ordinary least squares minimization technique is not always satisfactory since the estimates often have a low bias but a large variance. He came out with lasso [10] which shrinks or sets some coefficients to zero. Lasso stands for Least Absolute Shrinkage and Selection Operator. The idea has two goals: first to gain more interpretation by focussing on relevant predictors and, secondly to improve the prediction accuracy by reducing the variance of the predicted values. Lasso minimizes the L_2 loss function penalized by the L_1 norm on the parameters. This is a quadratic programming problem with linear inequality constraints and it is intractable when the vector of parameters is very large.

In literature, some efficient methods have been proposed to solve the exact Lasso namely the least angle regression by Efron *et al.* [1] and the homotopy method by Osborne *et al.* [6]. These methods were developed specifically to solve the least squares problem (i.e. using L_2 loss). They work well where the number of predictors is small. However, they are not adapted to nonparametric and classification tasks. To encounter these limitations, Zhao and Yu [12] proposed an algorithm with the name of BLasso (Boosted Lasso). The advantage of this algorithm lies in its ability to function with an infinite number of predictors and with various loss functions. These are some characteristics of the algorithms belonging to the boosting family.

The boosting mechanism was proposed by Schapire [8] in 1990. Since then, many algorithms have emerged [2, 3, 5, 9, 11] and boosting has become one of the most successful machine learning techniques. The underlying idea is to combine many weak classifiers (also called *hypotheses*) in order to obtain one final "strong" classifier. Boosting is an additive model which builds up one hypothesis after another by re-weighting the data for the next iteration—increasing the weights of misclassified images and decreasing those of well classified ones. This concept helps to generate different hypotheses, putting emphasis on misclassified examples, typically those located near the decision boundary in the feature space. Although it is an intuitive algorithm, boosting may overfit the training data, particularly when it runs for a large number of iterations T in high dimensional and noisy data [4, 7]. That's why, adding a regularization term is usually needed.

Unlike boosting, and in order to approximate lasso solutions, BLasso adds a backward step after each iteration of boosting. Thus, one is able to build up solutions with a coordinate descent manner and then take a look back at the consistency of these solutions regarding the model complexity. Added to that, BLasso does not suffer from the stop condition like an ordinary boosting does. In fact, and for a boosting procedure, fixing a large value of T implies a possible overfitting not to mention the long prediction time. On the other hand, setting T to a small value may lead to underfitting. Therefore, the model may be non-discriminant, inconsistent and might not cover the variability inside the category itself. A usual boosting algorithm can also be qualified as oblivious as it always functions in a forward manner aiming to minimize the empirical loss. Although the concept of re-weighting is interesting, at an iteration $t+1$, we have no idea whether the t previous generated hypotheses are good enough or not versus the model complexity.

3. LEARNING VISUAL PATCHES

3.1 Multiple-Instance Learning with BLasso

3.1.1 Training a Weak Classifier

Background information sometimes plays a primordial role in recognition. Thus, we propose to train our classifiers in a context of weak learning. That is, a training image is labeled as a whole sample. It will take the label $+1$ if it contains the object, -1 if not. This is also known as multiple instance learning. It deals with uncertainty of instance labels. An image is viewed as a bag of multiple features which are the local visual signatures. The bag will have only one label according to whether or not it includes at least one positive instance. It follows that it is only certain for a negative bag that there are no objects. Moreover, using a multi-instance approach has the luxury of unsupervised learning. It gives more freedom to the algorithm to select background features whenever they turn out to be useful to characterize the category.

Visual patches are viewed as weak classifiers. A weak classifier h_k represents a coordinate of base learners. Its weight is strictly positive if it was chosen at least once during the boosting process and remains zero if not. In the context of object categorization, the weak hypothesis h_k is a local image signature F_k with an optimal radius r_k (Opelt *et al.* [5]). In other words, h_k corresponds to a hypersphere centered on a local image feature F_k. For a test image x, h_k will output $+1$ if the distance between x and F_k is less than r_k and -1 otherwise. Discriminant radii of weak classifiers are optimal in the sense that the classification error is as minimal as possible. They are determined through a distance matrix. Each entry of the matrix represents a local feature with a corresponding ranked list of the training images. The distance between any feature F_{ki} belonging to the image I_i and any image I_j of the training set is defined by:

$$\mathrm{d}(F_{ki}, I_j) = \min_{1 \leq k' \leq M_j} \mathrm{d}(F_{ki}, F_{k'j})$$

Let $S_i = (I_i, l_i)$ be the couple representing the training image I_i labeled with $l_i \in \{-1, 1\}$ and w_i be the weight associated to I_i. $S = \{S_1, \cdots, S_N\}$ represents the set of all the training data. We denote by $d_{k,\eta}$ the distance separating the feature F_k from the image whose rank is η. The optimal radius r_k is determined after two steps. First we compute the curve corresponding to the sums of the weighted image

labels and take the index where the maximum is achieved:

$$\hat{\eta} = \arg \max_{1 \leq \eta \leq N} \sum_{i=1}^{\eta} w_i \cdot l_i \quad (1)$$

Then, r_k is given by:

$$r_k = \frac{d_{k,\hat{\eta}} + d_{k,\hat{\eta}+1}}{2} \quad (2)$$

Training a weak classifier is a simple boosting iteration. At an iteration t, we compute the score s_k corresponding to the feature F_k based on the image weights

$$s_k = \max_{m} \sum_{j=1}^{m} w_j \cdot l_j \quad (3)$$

Then, we select the feature which obtains the highest score. Note that Eq. (1) is in accordance with Eq. (3) in the sense that Eq. (1) looks for the index at which the score curve is maximum.

Using AdaBoost algorithm, Opelt *et al.* [5] considered an infinity of base learners. At the iteration t, and after selecting the best feature F_k, the base learner associated to F_k is defined by the couple (F_k, r_k) and is attributed a weight according to the training error. (r_k is given by eq. 2.) This seems plausible because AdaBoost uses a steepest descent to converge.

In our work, and for the sake of interpretability, each feature F_k represents only one base learner and should at most be selected once. It has to have its weight determined after several boosting iterations (forward stagewise). However, whenever selected, this base learner sees its radius r_k changes according to the weight updates, thus the final r_k attributed is the average of all the computed radii.

3.1.2 The Algorithm

Our algorithm is viewed as a boosting method constrained by a L_1-regularization term. L_1-regularization is equivalent to lasso [10]. Let $\beta = (\beta_1, \cdots, \beta_k, \cdots)^T$ be the vector of parameters to estimate (the weights of the weak hypotheses). β is initially zero. The lasso loss function Γ can be written as:

$$\Gamma(\beta, \lambda) = \sum_{n=1}^{N} L_2(S_n, \beta) + \lambda \cdot ||\beta||_1 \quad (4)$$

where $||\beta||_1 = \sum_j |\beta_j|$ denotes the L_1 norm of the vector β and $\lambda \geq 0$ is the parameter controlling the amount of regularization applied to the estimate. In order to get sparse solutions with an efficient shrinkage tradeoff, λ usually takes a moderate value since a large value may set these coefficients to exactly zero, leading to the null model. On the other hand, setting λ to zero reverses the lasso problem to minimizing the unregularized empirical loss. In our case, for classification, we replace L_2-loss function by the *exponential* loss function.

Since the exact lasso minimization is not tractable, BLasso [12] tries to find the same solutions as lasso with more cautious steps. Indeed it works with both forward and backward steps. Forward steps are used to minimize the empirical loss. On the other hand, backward steps minimize the regularization. In fact, at each iteration, a coordinate β_j is selected and updated by a small step size $\varepsilon > 0$. It has been shown that it is preferable to choose a very small step size so that BLasso can approximate the lasso path perfectly. In practice, ε should always be less than 0.1. In the original BLasso algorithm, forward steps can be either positive or negative (update by $\pm\varepsilon$). However, we chose not to. Our forward steps are always positive. We justify this choice by the fact that a selected classifier is a visual patch that contributes to build the object category model, thus it has to be positive.

Algorithm 1 BLasso

1. Initialization: $\beta = 0$
Make a forward step and initialize λ
2. Backward and forward steps:
Find the backward step that leads to the minimal empirical loss.
if the step decreases the lasso loss **then** take it.
else make a forward step and relax λ if necessary
3. Repeat step 2 until $\lambda \leq 0$.

Algorithm 1 gives a general overview on the BLasso mechanism. In our implementation, and in order to minimize the empirical loss, we used a weighting scheme as in AdaBoost. Adopting this strategy is appealing because it gives more attention to the misclassified observations by increasing their respective weights. Note that, at initialization, all weights are equal to $1/N$.

At the iteration t, a forward step can be summarized in the next five points.

1. Update weights

$$w_n^{(t+1)} = \frac{w_n^{(t)} \cdot \exp\big(-\varepsilon \cdot l_n \cdot h_\kappa^{(t)} I(n)\big)}{\tau} \quad (5)$$

τ is a normalization constant s.t. $\sum_{n=1}^{N} w_n^{(t+1)} = 1$

2. Train the classifier and get the best hypothesis $h_\kappa^{(t+1)}$
3. $\hat{\beta}^{(t+1)} = \hat{\beta}^{(t)} + \varepsilon \cdot \mathbf{1}_\kappa$
4. $$\lambda_{t+1} = \min\Big[\lambda_t, \frac{1}{\varepsilon}\big(\sum_{n=1}^{N} L(S_n, \hat{\beta}^{(t)} - \sum_{n=1}^{N} L(S_n, \hat{\beta}^{(t+1)} - \xi)\big)\Big] \quad (6)$$
5. $I_A^{(t+1)} = I_A^{(t)} \cup \{\kappa\}$ where I_A is the active index set.

Note that equation 5 is in accordance with the weight update mechanism used in AdaBoost. In fact, we just replaced the estimated hypothesis weight by the forward step ε.

The variable ξ, used when updating λ, is a tolerance parameter that is *strictly* positive. This parameter is added to gain more stability and it should be set to a very small value. Moreover, the initial value of λ can be obtained by the same formula (eq. 6) but with omitting ξ (as if $\xi = 0$). ξ is also used to decide whether to accept or reject a backward step. A backward step consists in finding the step that leads to the minimal empirical loss. It is given by:

$$\hat{j} = \arg \min_{j \in I_A^{(t)}} \sum_{n=1}^{N} L(S_n, \beta^{(t)} - \varepsilon \cdot \mathbf{1}_j) \quad (7)$$

This step is taken if and only if it decreases the lasso loss. That is, if $\Gamma(\beta^{(t)} - \varepsilon \cdot \mathbf{1}_{\hat{j}}, \lambda_t) - \Gamma(\beta^{(t)}, \lambda_t) \leq -\xi$ then

$$\beta^{(t+1)} = \beta^{(t)} - \varepsilon \cdot \mathbf{1}_{\hat{j}} \; ; \; \lambda_{t+1} = \lambda_t$$

3.2 Efficient Implementation

The inescapable processing burden for training is the computation of the distance matrix. To accelerate the process, one solution consists in trading accuracy for time by relying on approximate rather than exhaustive search. This could be achieved through range queries with an appropriate index structure like LSH for example. However, extreme care should be taken because it may drastically lower the quality of weak classifiers. This paper doesn't cover the gains obtained with such optimization.

The next processing burden and greediest block during training is computing the loss function. In fact, we need to compute both of the empirical and lasso losses many times during each iteration. The complexity increases every time a *new* coordinate is chosen. Luckily, the lasso loss can be deduced from the empirical loss. It follows that, in order to compute the current lasso loss and the backward lasso loss, we just need to compute their respective empirical losses. On the other hand, and at each iteration, only one coordinate is modified while all the other coordinates remain unchanged. Therefore, assuming that memory is cheaper than processing time, the computation of the empirical loss can be accelerated—at an iteration t—by preserving in memory the classification values of the previous state for each coordinate β_j and for each image I_n. Since our solutions are sparse, even when using a large number of training images, this method is still applicable. Each image I_n will have a storage vector ζ_n. To simplify the notation, we will consider that the j^{th} entry of the vector ζ_n (i.e. $\zeta_n(j)$) refers to the classification value of the variable β_j

$$\zeta_n(j) = \beta_j \cdot h_j(I_n) \qquad 1 \leq n \leq N; j \in I_A \tag{8}$$

Thus, when changing a hypothesis h_k in the next iteration, we only need to update (if β_k has already been selected before) or create new (if the index k is selected for the first time) N classification values. That is, we have to compute $\zeta_n(k)$ for all $1 \leq n \leq N$

The storage of the classification values is more important when searching for the backward step. The empirical loss is not computed just once but at least card$\{I_A\}$ times where card$\{I_A\}$ is the cardinal of the index set (It is computed card$\{I_A\}+1$ times if the coordinate is selected for the first time.) Given an image I_n, for each coordinate j, we need to compute the empirical loss based on the coordinates $\beta - \varepsilon \mathbf{1}_j$. Except for the coordinate β_j, the other classification values have already been computed and stored in $\zeta_n(k)$ with $k \neq j$. Moreover, the absolute difference between $\zeta_n(j)$ (already computed if the hypothesis is old) and the classification value δ_{nj} that we need to compute is ε

$$\text{if } \begin{cases} \zeta_n(j) < 0 & \Rightarrow \quad \delta_{nj} = \zeta_n(j) + \varepsilon \\ \zeta_n(j) > 0 & \Rightarrow \quad \delta_{nj} = \zeta_n(j) - \varepsilon \end{cases} \tag{9}$$

It follows that the classification value of the image I_n (i.e. $\sum_k \zeta_n(k)$) will only change by an absolute difference of ε. We denote by $\overline{\zeta}_n(j) = \sum_{k \neq j} \zeta_n(k) + \delta_{nj}$, that is:

$$\overline{\zeta}_n(j) = \begin{cases} \sum_k \zeta_n(k) + \varepsilon & \text{if } \zeta_n(j) < 0 \\ \sum_k \zeta_n(k) - \varepsilon & \text{if } \zeta_n(j) > 0 \end{cases} \tag{10}$$

This formulation helps to locate the backward step quickly. In fact, it only takes a linear time according to the number of the selected hypotheses. After subtracting the value ε from the coordinate β_j, the empirical loss E_j is computed as follows:

$$E_j = \sum_{n=1}^{N} \exp(-l_n \cdot \overline{\zeta}_n(j)) \tag{11}$$

The backward step $\hat{j}$ is then defined by

$$\hat{j} = \arg\min_j E_j \tag{12}$$

When the algorithm proceeds and selects a coordinate g at the iteration t, the stored values will be altered as follows. If g is selected for the first time, then

$$\forall n \qquad \overline{\zeta}_n(g) = \sum_{k \neq g} \zeta_n(k) \quad ; \quad \zeta_n(g) = \varepsilon \cdot h_g(I_n) \tag{13}$$

and

$$\overline{\zeta}_n^{(t)}(j) = \overline{\zeta}_n^{(t-1)}(j) + \varepsilon \cdot h_g(I_n) \qquad \forall j \neq g \tag{14}$$

Now, if g already belongs to the active index I_A, then

$$\zeta_n^{(t)}(g) = \zeta_n^{(t-1)}(g) + \text{sign}(\zeta_n^{(t-1)}(g)) \cdot \varepsilon \tag{15}$$

In this case, equation (14) is valid for all $j \in I_A$. Note that when $j = g$, this equation automatically takes into account the backward step (i.e the term $-\varepsilon \cdot \mathbf{1}_g$) because it was not added in the first place.

3.3 Prediction

For classification purpose, each image is predicted separately by looping through all weak classifiers. However, to be more efficient in retrieval, we predict by means of range queries. The final scores of all images are computed at the same time. First, these scores are initialized to zero. Then, and for each weak classifier, we query the search engine to retrieve all the images that fall into its hypersphere. Consequently, these images will see their scores incremented each by the weight of this weak classifier. On the other hand, images that are not returned by the system will have their scores decremented by the same weight of this classifier. After looping through all weak classifiers, each image ends up with a score judging its relevance to the concept.

The process described earlier can be relatively slow depending on the database size and the number of hypotheses constituting the model. For applications that have a fixed image database (not updated online), we can compute *a priori* the distances separating each weak classifier from each image and load them when the search engine starts. This could be achieved with no worries because the models used are concise and so won't consume too much computer memory. Moreover, since distances are computed offline, the process can be done either exhaustively or using an index structure. Unlike retrieval with range queries, to answer a given query, images are processed one by one. For each image, the distances to the model defined are obtained through a lookup table then the score is computed by comparing these distances to the corresponding classifier radii.

4. EXPERIMENTS

4.1 Performance Evaluation

Our experiments aims to prove that sparse models are still able to achieve good retrieval results. We first evaluated the retrieval effectiveness on 10 object categories belonging to

the ImageNet database[1]. It consists on comparing the performance between BLasso and AdaBoost. The name of the categories as well as the composition of the database used for training are given in table 1. Note that for each object category, we built the counter-class (negative samples) by randomly collecting images from the other categories while keeping the same number. We relied on SIFT descriptors as the basic image primitives. Moreover, each image contained at most 500 features. The original image set was divided equally between training and test. Therefore, the test database has the same composition shown in table 1 (a total of 6830 images). For all our experiments, the parameters of BLasso were tuned as follows

$$\xi = 10^{-6} \quad \text{and} \quad \varepsilon = \frac{1}{80}$$

This choice is made based on the experiments in [12]. For AdaBoost, we fixed the number of iterations $T = 100$.

For evaluation, we used the average precision. It is a reliable measure for retrieval and is computed as the area under the precision-recall curve. Precision is defined as the ratio of the number of correct answers to the number of the documents retrieved. Recall is defined as the ratio of the number of correct answers to the number of all the relevant documents in the database. Results are given in table 2.

camel	laptop	penguin	revolver	snail	sunflower	tennis_racket	tomato	watch	zebra
714	694	656	652	731	700	649	616	681	737

Table 1: Composition of the training set.

Category	# features			Average precision	
	Original	BL	AB	BL	AB
camel	282576	66	100	**0.2589**	0.2568
laptop	191445	69	100	**0.4606**	0.4467
penguin	264364	84	100	**0.3406**	0.3257
revolver	122472	2	30	0.2003	**0.3134**
snail	214597	44	100	0.261	**0.2784**
sunflower	264355	101	100	**0.5589**	0.5065
tennis_racket	173781	46	100	0.2287	**0.3156**
tomato	210641	62	100	**0.5424**	0.5244
watch	174818	51	100	0.413	**0.4712**
zebra	304673	78	100	**0.7146**	0.7029
Average	220372.2	60.3	93	0.3979	0.4142

Table 2: Performance evaluation of BLasso (BL) and AdaBoost (AB) in 10 categories of ImageNet.

BLasso outperformed AdaBoost in 6 categories (camel, laptop, penguin, sunflower, tomato and zebra). However, on average, AdaBoost was better approximately by 1.6%. This result is due to the clear domination of AdaBoost in the categories `revolver` and `tennis_racket` where the performance was respectively higher by 11.3% and 8.7%. On the other hand, it is noticeable that the feature selection performed by BLasso is better. In fact, the number of the features composing the model is fewer. Knowing that the prediction time is linear with the model size, the time needed

[1]http://www.image-net.org/

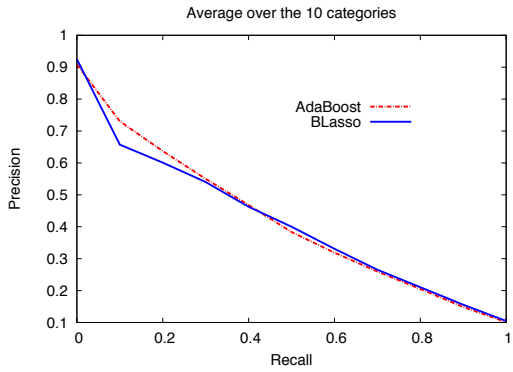


Figure 1: Precision-recall curves for BLasso and AdaBoost.

by AdaBoost is 1.5 times the time needed by BLasso. Furthermore, and in general, AdaBoost get stuck at the stop condition defined ($T = 100$) and didn't stop earlier except for `revolver`. This is problematic for AdaBoost because it proves that the solutions found are not statistically optimal, they might be visually correlated due to an excessive selection or they might need other features to be added in order to well cover the intra-class variability. Precision-recall curves are given in figure 1. They were averaged over the 10 categories. As proven numerically, the average performance of AdaBoost is slightly higher. However, we notice that for the first results returned (recall of 5%), the precision given by BLasso is similar or slightly better.

The second experiment aims to measure to which extent the models can be generic. In fact, the selection of ImageNet categories was done in purpose to be able to predict with the models generated on the same concepts belonging to Caltech256[2] database. Our test database was constructed from 15,360 images from all the 256 Caltech categories with only 586 images relevant to the 10 predicted categories. Details on the composition of these images are given in table 3. In this experiment, we compare the performance of the models trained with an image set collected from Caltech to the models trained with an image set collected from ImageNet. Note that the number of the training images used in the Caltech model is the same used in prediction (cf. table 3).

028.camel	127.laptop-101	158.penguin	172.revolver-101	189.snail	204.sunflower-101	218.tennis_racket	221.tomato	240.watch-101	250.zebra
55	64	75	50	60	40	41	52	101	48

Table 3: Composition of the training/test set.

Results of this experiment are presented in table 4. We notice that the models belonging to ImageNet outperformed the models of Caltech in 6 categories and that they also did better, on average. This proves that the models generated are generic in terms that, statistically, they don't overfit. Therefore, they can be applied to various test databases.

[2]http://www.vision.caltech.edu/Image_Datasets/Caltech256/

Category	Average precision	
	Caltech model	Image-net model
028.camel	0.0056	0.0132
127.laptop-101	0.0388	0.0382
158.penguin	0.01	0.0159
172.revolver-101	0.0128	0.0096
189.snail	0.0067	0.0178
204.sunflower-101	0.567	0.759
218.tennis-racket	0.0819	0.017
221.tomato	0.0156	0.1018
240.watch-101	0.0458	0.0792
250.zebra	0.0933	0.0787
Average	0.0878	0.113

Table 4: Illustration of prediction on a different database.

4.2 Interactive Retrieval

4.2.1 How It Works

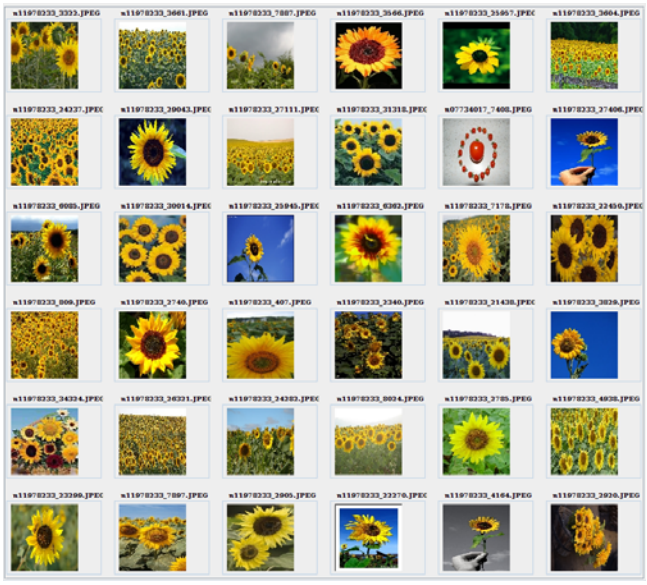
Interactive search is based on the user specialization of the models. Thanks to interpretability, the trained patches allow users to retrieve images, with not just the object inside, but also within a specific context, or with a desired scale or pose. For example, one may need to search for images containing cars in sand roads or airplanes in airports, etc.

Figure 2 is a snapshot of the user interface we developed. It gives an idea about what our interactive retrieval looks like. On the left side, users can choose a category from a list of predefined visual categories. The corresponding model is then loaded. Consequently, users can browse the visual patches displayed and start to form their own concept. In figure 2, the model of the sunflower category is chosen. In the left panel, we can browse all the 101 patches selected during training (cf. table 2). Note that the patches are displayed with their relative weights learned during training. However, when building their models, users have the possibility to change these confidence measures according to their own perception. That is, the more relevant a feature is, the higher the weight it will be assigned. Each visual patch chosen as well as its weight are displayed in the middle panel (11 patches in our snapshot). The visual query is now ready and the search engine can be interrogated. The retrieved images are displayed in the right panel.

4.2.2 Interactively Building Concepts

In this section, we will give visual illustrations on how interactivity is beneficial. The first example is given in figure 3. The user-specialized model used for retrieval (shown at the top) comprises six patches. The choice to make for displaying patches in a way that truly takes into account the description involved is still challenging. Here, we displayed the patches in gray-scale just to point out that the description is not color-related. They were taken from the sunflower model and they represent a global view of sunflowers within their context. Next, in fig. (3-2), we show the first page search results. As we can notice, the retrieved images match the query. They mostly contain sunflowers within a context (field, vegetation, sky). This result is due to the scale information brought by the patches. On the other hand, notice the presence of an image containing tomatoes. Even though it is not appreciated, such result explains the descriptor limitations. The image indeed looks like a sunflower. Now, have a look to the example shown in figure 2

(3-1) A specialized sunflower model used for retrieval.



(3-2) Search results.

Figure 3: Retrieving sunflowers.

and notice the difference between first ranked results. The model used is displayed in the middle panel. Unlike the previous example, the images retrieved here tend to occupy all the image area.

As a second example, we present two different queries on the zebra category and their respective results (cf. fig. 4 and 5). Both queries are related to the zebra category. The first query focus on the zebra upper-front part (head) while the second query is rather a general and global view of a zebra.

Our last experiment for interactivity is an attempt to build a hierarchical search engine that can semantically provide models for top level concepts and their sub-concepts. The idea is to draw together the categories `camel`, `penguin`, `snail` and `zebra` under the concept *animal*, then the categories `laptop`, `revolver`, `tennis_racket` and `watch` under the concept *man-made* and finally the categories `sunflower` and `tomato` under the concept *vegetation*. For training purpose, we used 30 images for each sub-concept. In summary, there were 120 images used in *animal* and *man-made* concepts and only 60 images used for *vegetation*. For prediction, we used a total of $13,363$ images. AP results are given in table 5. They are rather promising and demonstrate the feasibility of a hierarchical search engine with our method.

Category	# features	AP
animal	36	0.5516
man-made	22	0.5536
vegetation	33	0.4224

Table 5: Results of top semantical concepts.

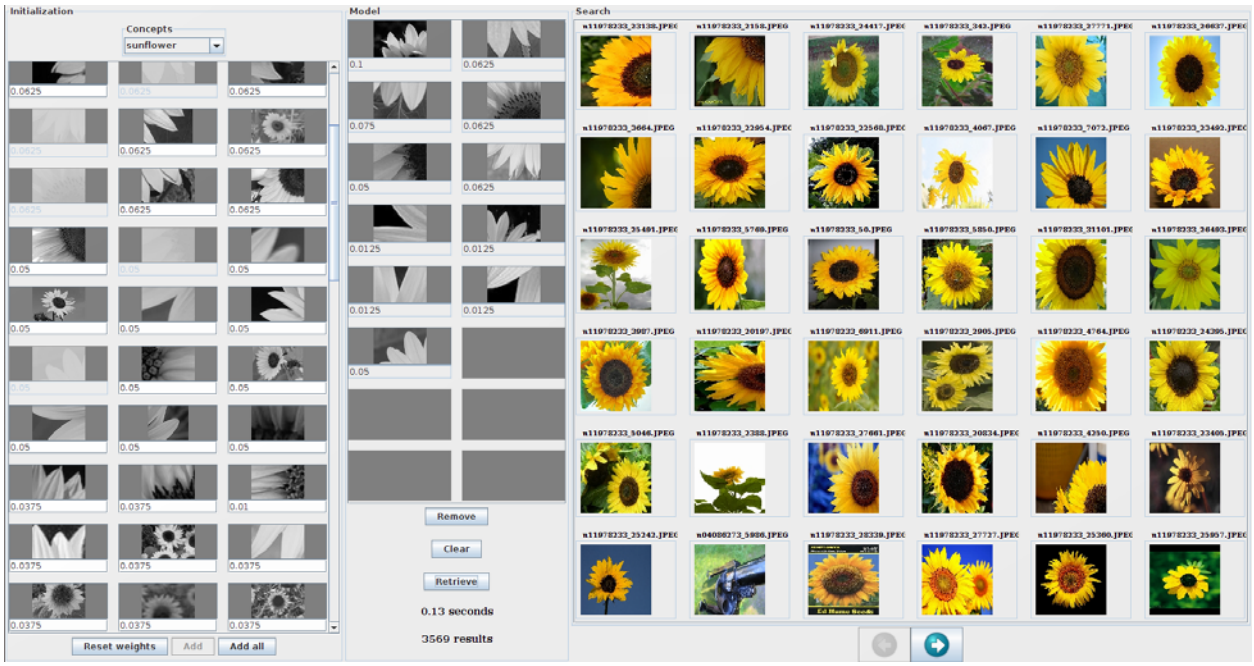
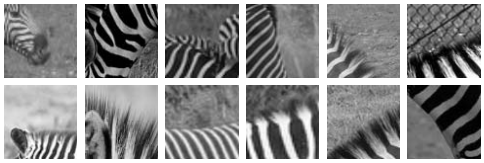
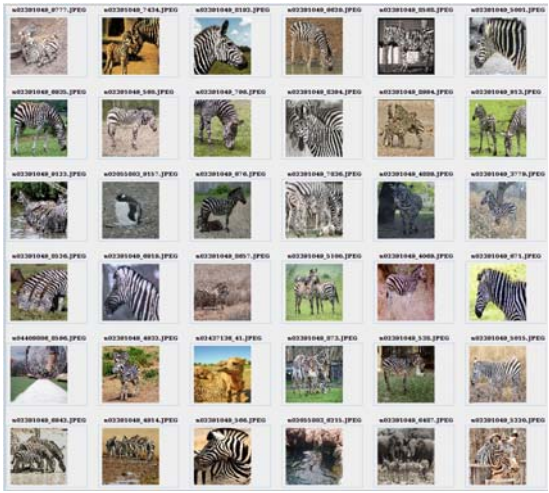


Figure 2: Example of an interactive search



(4-1) Zebra specialized model used for retrieval.



(4-2) Search results.

Figure 4: Example of patches to focus on the upper-front part.

(5-1) Zebra specialized model used for retrieval.

(5-2) Search results.

Figure 5: Example of patches to look for a global view of zebras.

Full model.

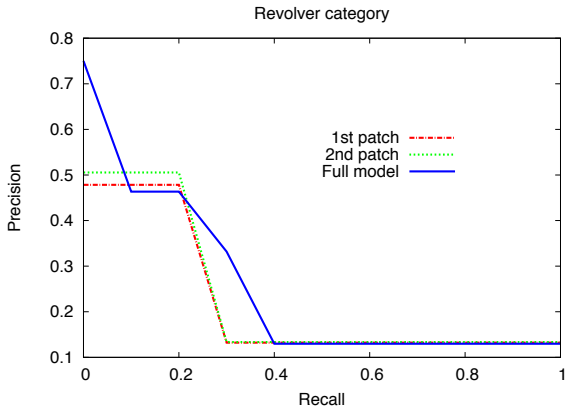


Figure 6: Performance of the revolver category.

4.3 Limitations

In our experiments, we noticed that it is possible for a single patch not to perform well if used by itself even if it is interpretable enough for a human. Two explanations can justify this limitation. The first possible cause is that our visual representation of the patch (gray-scale image corresponding to the description window of an interest point) doesn't faithfully reproduce the information the descriptor characterized. That is, the information we perceive from the patch is more complete than the semantic information coded by SIFT. The second explanation is that, statistically speaking, the database may be somehow skewed so that a single feature cannot be very discriminative. This problem is illustrated in figure 6 by the `revolver` category. The full model only comprises two patches that are shown at the top of the figure and their corresponding precision-recall curves are given below. Compare precision for the first ranked results (say 5% recall), we see that the full model behave quite well while each feature, alone, doesn't.

5. CONCLUSIONS

We introduced a new way to search for concepts in heterogeneous image databases through sparse and interpretable models. The models are a weighted set of visual patches that are directly mapped to image local features. For a given category, they represent the best features that summarize the intra-class variability. Consequently, we use them to provide a start page allowing users to build specialized models that, somehow, represent the concepts in their mind. Furthermore, we gave an efficient way to implement our learning strategy with the BLasso algorithm. Experiments revealed that our method gives equivalent performance than AdaBoost while consistently reducing the model complexity, giving better interpretability and saving the prediction time. Qualitatively, we showed how the interaction with the model can improve retrieved results.

In our future work, we plan to quantitatively evaluate user interactivity by generating a ground truth for more complex semantic queries. Moreover, we will think about experimenting new types of constraints in order to emphasize on some local aspects in images and see the subsequent effects on the model composition. We also intend to enrich patch description by using more coherent and compound representation through feature sets. Another remaining challenge is how to visually display the patches in a way to faithfully reproduce the underlaid description.

6. ACKNOWLEDGMENTS

This work was funded by the EU through the European Integrated Project GLOCAL http://www.glocal-project.eu/ and the Agropolis Foundation through the project Pl@ntNet http://www.plantnet-project.org/papyrus.php?langue=en.

7. REFERENCES

[1] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Annals of Statistics*, 32:407–499, 2004.

[2] Y. Freund and R. 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.

[3] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28, 2000.

[4] A. J. Grove and D. Schuurmans. Boosting in the limit: Maximizing the margin of learned ensembles. In *In Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 692–699, 1998.

[5] A. Opelt, M. Fussenegger, and P. Auer. Generic object recognition with boosting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(3):416–431, 2006. Member-Pinz„ Axel.

[6] M. R. Osborne, B. Presnell, and B. A. Turlach. On the lasso and its dual. *Journal of Computational and Graphical Statistics*, 9(2):pp. 319–337, 2000.

[7] G. Rätsch, T. Onoda, and K.-R. Müller. Soft margins for adaboost. *Mach. Learn.*, 42(3):287–320, 2001.

[8] R. E. Schapire. The strength of weak learnability. *Mach. Learn.*, 5(2):197–227, 1990.

[9] Y. Singer. Leveraged vector machines. In *Advances in Neural Information Processing Systems 12*, pages 610–616. MIT Press, 2000.

[10] R. Tibshirani. Regression shrinkage and selection via the lasso. *J. Roy. Statist. Soc. Ser. B*, 58(1):267–288, 1996.

[11] K. Tieu and P. Viola. Boosting image retrieval. *Int. J. Comput. Vision*, 56(1-2):17–36, 2004.

[12] P. Zhao and B. Yu. Stagewise lasso. *J. Mach. Learn. Res.*, 8:2701–2726, 2007.