Image Classification using Random Forests and Ferns

Anna Bosch
Computer Vision Group
University of Girona
aboschr@eia.udg.es

Andrew Zisserman
Dept. of Engineering Science
University of Oxford
az@robots.ox.ac.uk

Xavier Muñoz
Computer Vision Group
University of Girona
xmunoz@eia.udg.es

Abstract

We explore the problem of classifying images by the object categories they contain in the case of a large number of object categories. To this end we combine three ingredients: (i) shape and appearance representations that support spatial pyramid matching over a region of interest. This generalizes the representation of Lazebnik et al [16] from an image to a region of interest (ROI), and from appearance (visual words) alone to appearance and local shape (edge distributions). (ii) automatic selection of the regions of interest in training. This provides a method of inhibiting background clutter and adding invariance to the object instance's position, and (iii) the use of random forests (and random ferns) as a multi-way classifier. The advantage of such classifiers (over multi-way SVM for example) is the ease of training and testing.

*Results are reported for classification of the Caltech-*101 *and Caltech-*256 *data sets. We compare the performance of the random forest/ferns classifier with a benchmark multi-way SVM classifier. It is shown that selecting the ROI adds about* 5% *to the performance and, together with the other improvements, the result is about a* 10% *improvement over the state of the art for Caltech-*256*.*

1. Introduction

The objective of this work is image classification – classifying an image by the object category that it contains. This problem has been the subject of many recent papers [14, 15, 16, 26, 27] using the Pascal Visual Object Classes datasets or the Caltech-101 dataset. The release of challenging data sets with ever increasing numbers of object categories, such as Caltech-256 [13], is forcing the development of image representations that can cope with multiple classes and of algorithms that are efficient in training and testing.

To date the two approaches that have achieved the best performance on the smaller Caltech-101 dataset have involved an improved scene representation – the spatial pyramid matching of Lazebnik *et al.* [16], and an improved classifier – the SVM-KNN algorithm of Zhang *et al.* [26]. In this paper we build on both of these ideas.

First the image representation: [16] argued that Caltech-101 was essentially a scene matching problem so an image based representation was suitable. Their representation added the idea of flexible scene *correspondence* to the bag-of-visual-word representations that have recently been used for image classification [8, 22, 27]. We improve on their representation in two ways. First, for training sets that are not as constrained in pose as Caltech-101 or that have significant background clutter, treating image classification as scene matching is not sufficient. Instead it is necessary to "home in" on the object instance in order to learn its visual description. To this end we automatically learn a Region Of Interest (ROI) in each of the training images in the manner of [7]. The idea is that between a subset of the training images for a particular class there will be regions with high visual similarity (the object instances). These regions can be identified from the clutter by measuring similarity using the spatial pyramid representation of [16], but here defined over a ROI rather than over the entire image. The result is that "clean" visual exemplars [3] are obtained from the pose varying and cluttered training images.

The second extension over [16] is to represent both the appearance (using dense vector quantized SIFT descriptors, as they did) but also local shape (using a distribution over edges [5, 7]). These representations are common over all classes. As in Bosch *et al.* [5] the novelty is that the classifier has the ability to choose the weight given to shape, appearance and the levels of the pyramids. This facilitates representations and classification suited to the class. For example, a car can be better distinguished by its shape and a lion by its appearance, and not all objects require the same pyramid level, e.g. objects with higher intra-class spatial variation are best represented by lower pyramid levels.

Turning to the classifier, we employ here a random forest classifier. These classifiers were first introduced in [1] and developed further in [6]. Their recent popularity is largely due to the tracking application of [17]. They have been ap-

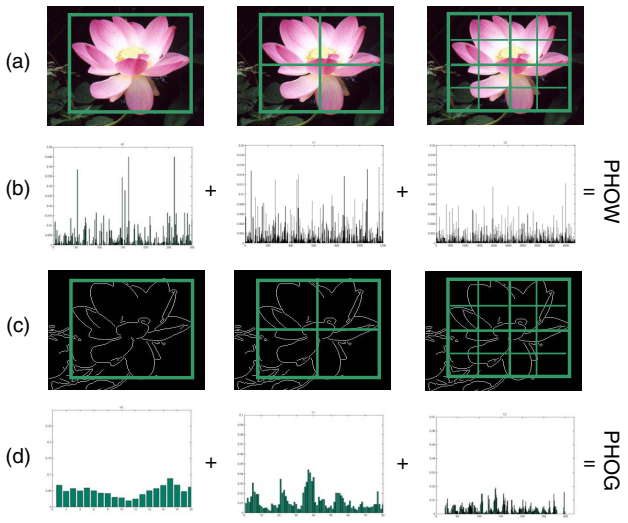

Figure 1. Appearance and shape spatial representation. (a,c) Grids for levels $l = 0$ to $l = 2$ for appearance and shape representation; (b,d) appearance and shape histogram representations corresponding to each level.

plied to object recognition in [20, 24] but only for a relatively small number of classes. Here we increase the number of object categories by an order of magnitude (from 10 to 256). The research question is how to choose the node tests so that they are suited to spatial pyramid representations and matching. The advantage of randomized trees, as has been noted by previous authors [25], is that they are much faster in training and testing than traditional classifiers (such as an SVM). They also enable different cues (such as appearance and shape) to be "effortlessly combined" [24].

We briefly give an overview of the descriptors and how the images are represented in § 2. Then in § 3 we describe how the ROIs are automatically learnt, and how they are used together with random forests (and ferns) to train a classifier. A description of datasets and the experimental evaluation procedure is given in § 4. Implementation details are given in § 5. § 6 reports the performance on Caltech-101 and Caltech-256, as well as a comparison with the state of the art. The paper concludes with a discussion and conclusions in § 7.

2. Image Representation and Matching

An image is represented using the spatial pyramid scheme proposed by Lazebnik *et al.* [16], which is based on spatial pyramid matching [12], but here applied to both appearance and shape. The representation is illustrated in Fig. 1. For clarity, we will describe here the representation of the spatial layout of an entire image, and then in section 3 this representation is applied to a ROI.

Appearance. We follow the approach of Bosch *et al.* [4]. SIFT descriptors [19] are computed at points on a regular grid with spacing M pixels. At each grid point the descriptors are computed over four circular support patches with different radii, consequently each point is represented by four SIFT descriptors. Multiple descriptors are computed to allow for scale variation between images. The dense features are vector quantized into V *visual words* [23] using K-means clustering. Implementation details are given in section 5

Shape. Local shape is represented by a histogram of edge orientations gradients (HOG [9]) within an image subregion quantized into K bins. Each bin in the histogram represents the number of edges that have orientations within a certain angular range. This representation can be compared to the traditional "bag of (visual) words", where here each visual word is a quantization on edge orientations. Again, implementation details are given in section 5.

Spatial pyramid representation. Using the above appearance and shape descriptors together with the image spatial layout we obtain two representations: (i) a Pyramid Histogram Of visual Words (PHOW) descriptor for appearance and (ii) a Pyramid HOG (PHOG) descriptor for shape [5, 7]. In forming the pyramid the grid at level l has 2^l cells along each dimension. Consequently, level 0 is represented by a N-vector corresponding to the N bins of the histogram (where $N = V$ for appearance and $N = K$ for shape), level 1 by a $4N$-vector etc, and the Pyramid descriptor of the entire image (PHOW, PHOG) is a vector with dimensionality $N \sum_{l=0}^{L} 4^l$. For example, for shape information, levels up to $L = 1$ and $K = 20$ bins it will be a 100-vector. In the implementation we limit the number of levels to $L = 3$ to prevent over fitting.

Image matching. The similarity between a pair of images I and J is computed using a kernel function between their PHOG (or PHOW) histogram descriptors D_I and D_J, with appropriate weightings for each level of the pyramid:

$$K(D_I, D_J) = exp\{\frac{1}{\beta} \sum_{l \epsilon L} \alpha_l d_l(D_I, D_J)\} \qquad (1)$$

where β is the average of $\sum_{l=0}^{L} \alpha_l d_l(D_I, D_J)$ over the training data, α_l is the weight at level l and d_l the distance between D_I and D_J at pyramid level l computed using χ^2 [27] on the normalized histograms at that level.

3. Learning the model

This section describes the two components of the model: the selection of the ROI in the training images, and the random forest/fern classifier trained on the ROI descriptors. It is not feasible to optimize the classification performance over both the classifiers and the selection of the ROI, so instead a sub-optimal strategy is adopted where the PHOG and PHOW descriptors are first used to determine the ROIs

of the training images, and then the classifier is trained on these fixed regions.

3.1. Selecting the regions of interest (ROI)

Caltech-256 (and several other datasets used for object recognition, such as PASCAL) has a significant variation in the position of the object instances within images of the same category, and also different background clutter between images (see Fig. 2). Instead of using the entire image to learn the model, an alternative is to focus on the object instance in order to learn its visual description. To this end we describe here a method of automatically learning a rectangular ROI in each of the training images. The intuition is that between a subset of the training images for a particular class there will be regions with high visual similarity (the object instances). It is a subset due to the variability in the training images – one instance may only be similar to a few others, not to all the other training images. These "corresponding" regions can be identified from the clutter by measuring their similarity using the image representation described in section 2 but here defined over a ROI rather than over the entire image.

Suppose we know the ROI r_i in image i and the subset of s other images j that have "corresponding" object instances amongst the set of training images for that class. Then we could determine the corresponding ROIs r_j of images j by optimizing the following cost function:

$$\mathcal{L}_i = \max_{\{r_j\}} \sum_{j=1}^{s} K(D(r_i), D(r_j)) \qquad (2)$$

where $D(r_i)$ and $D(r_j)$ the descriptors for the ROIs r_i and r_j respectively, and their similarity is measured using the kernel defined by (1). Here we use a descriptor formed by concatenating the PHOG and PHOW vectors. As we do not know r_i or the subset of other images we also need to search over these, i.e. over all rectangles r_i and all subsets of size s (not containing i). This is too expensive to optimize exhaustively, so we find a sub-optimal solution by alternation: for each image i, fix r_j for all other images and search over all subsets of size s and in image i search over all regions r_i. Then cycle through each image i in turn. The value for the parameter s depends on the intra-class variation and we explore its affect on performance in section 6.

In practice this sub-optimal scheme produces useful ROIs and leads to an improvement in classification performance when the model is learnt from the ROI in each training image. Fig. 2 shows examples of the learnt ROIs for a number of classes.

3.2. Random forests classifier

A random forest multi-way classifier consists of a number of trees, with each tree grown using some form of randomization. The leaf nodes of each tree are labeled by estimates of the posterior distribution over the image classes. Each internal node contains a test that best splits the space of data to be classified. An image is classified by sending it down every tree and aggregating the reached leaf distributions. Randomness can be injected at two points during training: in subsampling the training data so that each tree is grown using a different subset; and in selecting the node tests.

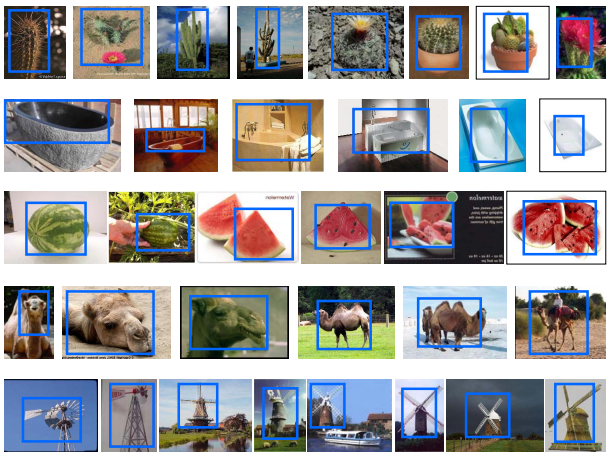

Figure 2. Automatic ROI detection. Examples from Caltech-256 for $s = 3$ for cactus, bathtub, watermelon, camel and windmill.

Growing the trees. The trees here are binary and are constructed in a top-down manner. The binary test at each node can be chosen in one of two ways: (i) randomly, i.e. data independent; or (ii) by a greedy algorithm which picks the test that best separates the given training examples. "Best" here is measured by the information gain

$$\Delta E = -\Sigma_i \frac{\mid Q_i \mid}{\mid Q \mid} E(Q_i) \qquad (3)$$

caused by partitioning the set Q of examples into two subsets Q_i according the given test. Here $E(q)$ is the entropy $-\sum_{j=1}^{N} p_j log_2(p_j)$ with p_j the proportion of examples in q belonging to class j, and $\mid \ . \ \mid$ the size of the set. The process of selecting a test is repeated for each nonterminal node, using only the training examples falling in that node. The recursion is stopped when the node receives too few examples, or when it reaches a given depth.

Learning posteriors. Suppose that T is the set of all trees, C is the set of all classes and L is the set of all leaves for a given tree. During the training stage the posterior probabilities $(P_{t,l}(Y(I) = c))$ for each class $c \in C$ at each leaf node $l \in L$, are found for each tree $t \in T$. These probabilities are calculated as the ratio of the number of images I of class c that reach l to the total number of images that reach l. Y(I) is the class-label c for image I.

Classification. The test image is passed down each random

tree until it reaches a leaf node. All the posterior probabilities are then averaged and the $\arg\max$ is taken as the classification of the input image.

3.3. Node tests for PHOG and PHOW

Recent implementations of random forests [17, 24] have used quite simple pixel level tests at the nodes (for reasons of speed). Here we want to design a test that is suitable for the representations of shape, appearance and pyramid spatial correspondence, that we have at our disposal. We also use a relatively simple test – a linear classifier on the feature vector – but also include feature selection at the test level. The tests are represented as:

$$T = \begin{cases} if\ \mathbf{n}^\mathsf{T}\mathbf{x} + b \leq 0 & \text{go to the right child} \\ otherwise & \text{go to the left child} \end{cases}$$

where $\mathbf{n}$ is a vector with the same dimension as the data vector $\mathbf{x}$. A node test is obtained by choosing a random number of features n_f, choosing a random n_f indexes, and filling those components of $\mathbf{n}$ with random numbers in the range $[-1, 1]$ (the remaining components are zero). The value of b is obtained as a random number as well. Although this is a simple linear classifier we will demonstrate in section 6 that it achieves very similar performances when compared with a multi-way SVM (M-SVM) and it considerably improves the speed.

Descriptor selection. We wish to enable the classifier to be selective for shape or appearance or pyramid level – since some classes may be better represented by each of these. For example, airplanes by their shape, tiger by its appearance, classes with high intra-class variation by lower pyramid levels, etc.

To combine the features, a test randomly selects the descriptor (shape or appearance). This is the same as giving weight 1 to one descriptor and weight 0 to the others. In both cases only one descriptor is used. In a similar manner pyramid levels are selected. A test randomly selects a level l, and only the indexes corresponding to this level are non-zero in $\mathbf{n}$.

An alternative way to merge the descriptors is to build a forest for each descriptor (eg. 50 trees using only shape information and 50 trees using only appearance information) and merge them (using the 100 trees) for the classification.

3.4. Random ferns classifier

To increase the speed of the random forest Ozuysal *et al.* [21] proposed random *ferns* classifiers. Ferns are non-hierarchical structures where each one consists of a set of binary tests (which in our case is 0 if $\mathbf{n}^\mathsf{T}\mathbf{x} + b > 0$ or 1 if $\mathbf{n}^\mathsf{T}\mathbf{x} + b \leq 0$). During training there are an ordered set of tests S applied to the whole training data set. This is in contrast to random forests where only the data that falls in a child is taken into account in the test.

As in random forests "leaves" store the posterior probabilities. During testing the probability that an image belongs to any one of the classes that have been learned during training is returned. The result of each test and the ordering on the set defines a binary code for accessing the "leaf" node.

As in random forests, the test image is passed down all the randomized ferns. Each node in the fern provides a result for the binary test which is used to access the leaf which contains the posterior probability. The posteriors are combined over the ferns in the same way as for random forests over trees.

3.5. Image Classification

For the test images a "sliding window" over a range of translations and scales is applied. A new sub-image W_I classified by considering the average of the probabilities $P_{t,l}(Y(I) = c)$:

$$\hat{Y}(I) = \arg\max_c \frac{1}{T}\Sigma_{t=1}^{T} P_{t,l}(Y(I) = c) \qquad (4)$$

where l is the leaf reached by image I in tree t. We classify an image I as the class C_k provided by the ROI which gives highest probability.

4. Datasets and Experimental Protocol

Caltech-101. This dataset (collected by Fei-Fei *et al.* [10]) consists of images from 101 object categories, and contains from 31 to 800 images per category. Most images are medium resolution, about 300×300 pixels. The significance of this database is its large inter-class variability.

Caltech-256. This data set (collected by Griffin *et al.* [13]) consists of images from 256 object categories and is an extension of Caltech-101. It contains from 80 to 827 images per category. The total number of images is 30608. The significance of this database is its large inter-class variability, as well as a larger intra-class variability than in Caltech-101. Moreover there is no alignment amongst the object categories. Fig. 3 shows some images from this dataset.

Experiments. Following standard procedures, the Caltech-101 data is split into 30 training images (chosen randomly) per category and 50 for testing – disjoint from the training images. For Caltech-256, 30 images are used for training and 25 for testing. For a comparison with [13] for Caltech-256, we report experiments without the last 6 categories and without clutter, this is 250 categories. The final performance score is computed as the mean recognition rate per class. The classification process is repeated 10 times, (changing the training and test sets), and the average performance score and its standard deviation are reported.

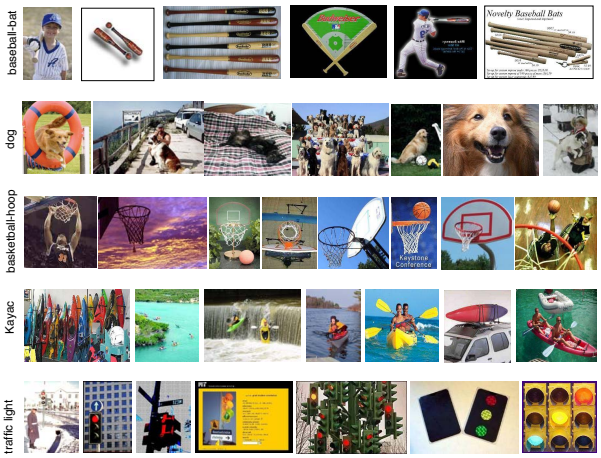

Figure 3. Some images from the Caltech-256 dataset.

no optimization.	$s = 1$	$s = 2$	$s = 3$	$s = 4$
38.7	42.5	42.9	43.5	42.8
± 1.3	± 1.0	± 1.0	± 1.1	± 1.0

Table 1. Caltech-256 performance when using 100 randomized trees with D=20, entropy optimization and all descriptors. The first column is without ROI optimization and the rest are for ROI optimization from $s = 1$ to 4 images. The standard deviation is given below each row.

5. Implementation

Appearance. For the appearance representation both grey level and colour cues are used (termed App_{Gray} and App_{Colour} respectively). SIFT descriptors are computed at points on a regular grid with spacing M pixels, here $M = 10$. At each grid point the descriptors are computed over circular support patches with radii $r = 4, 8, 12$ and 16 pixels. The patches with radii 4 do not overlap and the other radii do. For App_{Colour} the SIFT descriptors are computed for each HSV component. This gives a 128×3 D-SIFT descriptor for each point. In the case of App_{Gray}, SIFT descriptors are computed over the gray image (with intensity I = 0.3R + 0.59G + 0.11B) and the resulting SIFT descriptor is a 128 vector. Note that the descriptors are rotation invariant. The K-means clustering is performed over 5 training images per category selected at random. A vocabulary of $V = 300$ words is used here.

Shape. Edge contours are extracted using the Canny edge detector. The orientation gradients are then computed using a 3×3 Sobel mask without Gaussian smoothing. It has been shown previously [9] that smoothing the image significantly decreases classification performance. In the experiments two shape descriptors are used: one with orientations in the range $[0, 180]$ (where the contrast sign of the gradient is ignored) and the other with range $[0, 360]$ using all orientation as in the original SIFT descriptor [19]. We refer to these as $Shape_{180}$ and $Shape_{360}$ respectively. The histogram descriptor is discretized into $K = 20$ and $K = 40$ bins for for $Shape_{180}$ and $Shape_{360}$ respectively.

ROI detection. The optimization process is done by testing the similarity between a number of images s ranging from 1 to 4. The search is over the four parameters specifying the coordinates of the rectangle: x_{min}, x_{max}, y_{min} and y_{max}. The search is carried out over a translation grid with 10 pixel steps. The optimization is initialized with the ROI corresponding to the entire image, and then scaling the four parameters in steps of 0.1. The new ROI obtained after the scaling process is translated over the whole image and then scaled again. At each iteration we optimize the cost function (2) for each training image. The optimization terminates when there are no more changes in the ROIs or when the number of iteration reaches 10. For the descriptor we use the PHOG and PHOW vectors concatenated.

Randomized trees and ferns. At a given node, n_f features are randomly selected (the number used is discussed in section 6). The vector $\mathbf{n}$ is initialized with zeros and the n_f variables chosen are coefficients that are uniform random numbers on $[-1, 1]$. b is randomly chosen between 0 and the distance of the further point $\mathbf{x}$ from the origin. We then recursively build the trees by trying r different tests at each node and keeping the best one according to the entropy criterion of (3). As in [17], for the root node we chose $r = 10$, a very small number, to reduce the correlation between the resulting trees. For all other nodes, we used $r = 100$D, where D is the depth of the node. When choosing a binary test randomness is injected into the training set per tree: one third of the training images per category are randomly selected and used to determine the node tests by the entropy criterion, and the remaining training images are used to estimate the posterior probabilities in the terminal nodes. This heuristic involves randomizing over both tests and training data. When using the simpler approach (i.e. without using the criterion (3)), trees are grown by randomly selecting $\mathbf{n}$ and b without measuring the gain of each test, and all the training images are used to estimate the posterior probabilities. For the two methods, trees are grown until a maximal depth is reached or until less than 10 instances fall in the node. We test trees for D = 10, 15 and 20. To grow the ferns $r = 10$ is used for each binary test.

6. Image Classification Results

We first study the influence of different parameters using Caltech-256 as our test set. Then, in section 6.1 we compare the random forests with a multi-way SVM classifier for Caltech-101 and in section 6.2 we provide a comparison with the state-of-art. For the experiments the following parameters are used unless stated otherwise: 100 randomized trees with D=20, entropy optimization, and all the descriptors. Parameter optimization is carried out on a validation set (a sub-set of the training set, disjoint from the test set).

	Randomized Forests				
	Shp_{180}	Shp_{360}	App_C	App_G	All
RT	38.5	39.3	35.2	39.3	41.9
	± 0.8	± 0.9	± 0.9	± 1.0	± 1.2
EO	39.2	40.5	36.5	40.7	43.5
	± 0.8	± 0.9	± 0.8	± 0.9	± 1.1
	Randomized Ferns				
RT	37.7	38.1	34.7	38.9	41.0
	± 0.8	± 0.8	± 0.9	± 0.9	± 0.9
EO	38.9	39.7	36.5	39.2	42.6
	± 0.8	± 0.9	± 0.9	± 0.8	± 1.0

Table 2. Caltech-256 performance using appearance, shape and all the feature descriptors with randomized trees and ferns. 100 trees/ferns with D/S=20 and ROI with $s = 3$ are used. RT = Random Test, EO = Entropy Optimization. The standard deviation is given below each row.

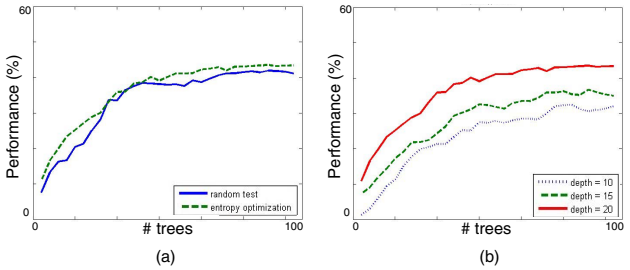

Figure 4. (a) Comparing the classification rates obtained using trees (D=20) grown by selecting tests that maximize the information gain (green dashed line) and by randomly chosen tests (blue solid line), as a function of the number of trees. Note that the performance saturates when more than 90 trees are used. (b) Comparing the classification rates obtained using trees with entropy optimization for D=10, 15 and 20 again as a function of the number of trees. All the descriptors are used in both graphs.

ROI. Table 1 shows the performances when changing the number s of images to optimize in the cost function. Without the optimization process the performance is 38.7%, and with the optimization this increases by 5%. There is not much difference between using 1 to 4 images to compute the similarity.

Node tests. The first two rows in Table 2 compare the performances achieved using a random forests classifier with random node test (first row) and with entropy optimization (second row) when using $Shape_{180}$, $Shape_{360}$, App_{Colour}, App_{Gray} and when merging them. Slightly better results are obtained for the entropy optimization (around 1.5%). When merging all the descriptors with entropy optimization performance for random forests is 43.5%. Taking the tests at random usually results in a small loss of reliability but considerably reduces the learning time. The time required to grow the trees drops from 20 hours for entropy optimization to 7 hours for random tests on a 1.7 GHz machine and Matlab implementation. Fig. 4a shows how the classification performance grows with the number of trees when using all the descriptors.

Forests vs ferns. The last two rows in Table 2 are performances when using random ferns. Performances are less than 1% worse than those of random forests (42.6% when using all the descriptors to grow the ferns). When using the random forest the standard deviation is small (around 1%) independent of the number of trees used. This is in contrast to random ferns where if few ferns are used the variance is very large, however as the number of ferns is increased the random selection method does not cause large variations on the classifier performance [21]. The standard deviation when fixing the training and test sets and training the trees/ferns 10 times is 0.3%. The main advantage of using ferns is that the training time increases linearly with the number of tests S, while for random forests it increases exponentially with the depth D. Training the ferns takes $1.5h$ with random splits and $4h$ with entropy optimization when using 100 ferns $S = 20$ and 25 training images per category. For both, random forests and ferns the test time increases linearly with the number of trees/ferns.

Number of trees/ferns and their depth. Fig. 4b shows performances, as the number of trees increases, when varying the depth of trees from 10 to 20. Performance is 32.4% for D=10, 36.6% for D=15 and 43.5% for D=20. When using ferns performances are 30.3%, 35.5% and 42.6% for S=10, 15 and 20 respectively. Increasing the depth increases performance however it also increase the memory required to store trees and ferns, as mentioned above.

When building the forests we experimented with assigning lower pyramid levels to higher nodes in the tree and higher pyramid levels to the bottom nodes. For example for a tree with D=20, pyramid level $l = 0$ is used until depth $d = 5$, $l = 1$ from $d = 6$ to $d = 10$, $l = 2$ from $d = 11$ to $d = 15$, and $l = 3$ for the rest. In this case performance decreases 0.2%. When merging forests by growing 25 trees for each descriptor (100 trees in total) and merging the probability distributions when classifying the performance increases 0.1%.

Number of features. All results above are when using a random number of features to fill the vector $\mathbf{n}$ in the linear classifier. Here we investigate how important the number of non-zero elements is. Fig. 5a shows a graph, for both random and entropy tests, when increasing the number of features used to split at each node. The number of features is increased from 1 to m where m is the dimension of the vector descriptor $\mathbf{x}$ and $\mathbf{n}$ for a level. It can be seen that the procedure is not overly sensitive to the number of features used, as was also demonstrated in [6]. Very similar results are obtained using a single randomly chosen input variable to split on at each node, or using all the variables. The standard deviation is decreased as we increase the number of features used and it is higher when using a random split.

Training data. The blue dashed line in Fig. 5b shows how the performance changes when increasing the number of training images from 5 to 30. Performance increases (by 20%) when using more training data meaning that with less data, the training set is not large enough to estimate the full posterior [2]. Since we have a limited number of training images per category and, as noted in the graph, performance increases if we increase the training data, we populate the training set of positive samples by synthetically generating additional training examples [15]. Given the ROI, for each training image we generate similar ROIs by perturbing the position ($[-20, 20]$ pixels in both x and y directions), the size of original ROIs (scale ranging between $[-0.2, 0.2]$) and the rotation ($[-5, 5]$ degrees). We treat the generated ROIs as new annotations and populate the training set of positive samples. We generate 10 new images for each original training example obtaining 300 additional images per category (resulting in a total of 330 per category). When using 5 training images plus the extra data the performance increases from 19.7% to 29.1%, and it increases form 43.5% to 45.3% when using 30 training images. The green line in Fig. 5b shows the performance when using extra training data.

6.1. Random Forests vs Multi-way SVM

In this section we compare the random forest classifier to the multiple kernel SVM classifier of [5] on Caltech-101 for the case of 30 training and 50 testing images. For training we first learn the ROIs using $s = 3$ and generate 10 new images for each original training image, as above.

In this case PHOW and PHOG (see Section 2) are the feature vectors for an M-SVM classifier, using the spatial pyramid kernel (1). For merging features the following kernel is used:

$$K(x, y) = \alpha K_A(x_{App}, y_{App}) + \beta K_S(x_{Shp}, y_{Shp}) \quad (5)$$

K_A and K_S is the kernel defined in (1). The weights α and β in (5) as well as the the pyramid level weights α_l in (1) are learnt for each class separately by optimizing classification performance for that class on a validation set using one vs the rest classification [5].

Performance for the random forests is 80.0% and with the M-SVM is 81.3%. However, using random forests/ferns instead of a M-SVM, is far less computationally expensive – the classification time is reduced by a factor of 40.

For Caltech-101 adding the ROI optimization does not increase performance as much as in the case of Caltech-256. This is because Caltech-101 has less pose variation within the training images for a class.

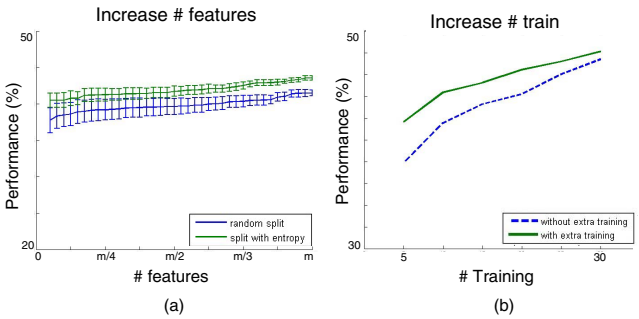

Figure 5. (a) performance when increasing the n_f to use at each node. The error bars show $\pm$ one standard deviation; (b) performance when increasing the number of training images with and without extra training data. 100 trees, D=20 and all the descriptors are used in both graphs. Entropy optimization is used in (b).

Dataset	C-101		C-256	
N_{Train}	15	30	15	30
M-SVM	–	81.3	–	–
	–	$\pm$0.8	–	–
R. Forests	70.4	80.0	38.6	45.3
	$\pm$0.7	$\pm$0.6	$\pm$0.6	$\pm$0.8
R. Ferns	70.0	79.2	37.5	44.0
	$\pm$0.7	$\pm$0.6	$\pm$0.8	$\pm$0.7
[5]	67.4	77.8	–	–
[13]	59.0	67.6	29.0	34.1
[26]	59.0	66.2	–	–
[11]	60.3	66.0	–	–
[16]	56.4	64.6	–	–
[18]	59.9	–	–	–

Table 3. Caltech-101 and Caltech-256 performances when 15 and 30 training images are used.

6.2. Comparison with the state of the art

Following the standard procedures for Caltech datasets we randomly select N_{train} = 5, 10, 15, 20, 25, 30, 40, $N_{test} = 50$ for Caltech-101 and $N_{test} = 25$ for Caltech-256. Results are obtained using 100 trees/ferns with D/S=20 and entropy optimization to split each node, ROI optimization, and increased training data by generating 300 extra images per category.

Table 3 summarizes our results and the state of the art for Caltech-101 and Caltech-256 when 15 and 30 training images are used. Fig. 6 shows our results and the results obtained by others as the number of training images is varied.

Caltech-256. Griffin *et al* [13] achieve a performance of 34.1% using the PHOW descriptors with a pyramid kernel (1) and M-SVM. Our performance when merging different descriptors with random forests is 45.3%, outperforming the state-of-art by 11%. All the above results are for 250 categories. For the whole Caltech-256 (not clutter) performance is 44.0%.

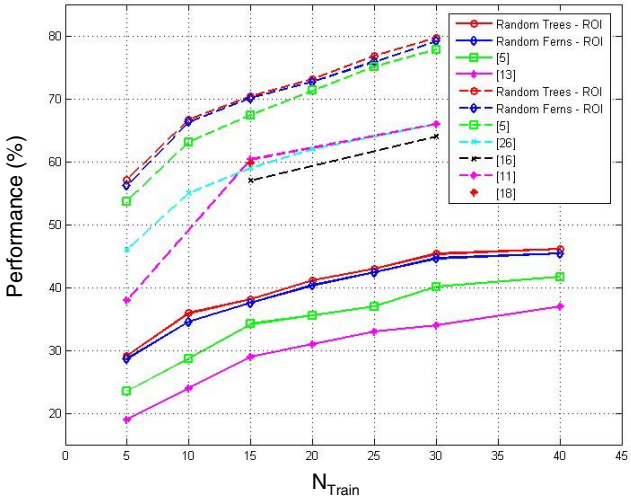

Figure 6. Performance as a function of the number of training images for Caltech-101 and Caltech-256. We compare our implementation with Random Forests/Ferns with the published results.

7. Conclusions

We have demonstrated that using random forests/ferns with an appropriate node test reduces training and testing costs significantly over a multi-way SVM, and has comparable performance.

We have improved on the state of the art for Caltech-101 and Caltech-256 and, as a summary, quantify approximately the contributions arising from each of the principal improvements over [5]: (i) Using the ROI detection and sliding window is a significant benefit. It increases performance from 3% to 5% depending on the degree of object pose variation within the datasets. (ii) generating extra data during training increases performance by 2%.

Future work will include more robust edge features and ROI detection using a more flexible region than a rectangle.

Acknowledgements

This work was partially funded by the University of Girona grant BR03/01, by EU Project CLASS, and by a MURI grant.

References

[1] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural Computation*, 9:1545–1588, 1997.

[2] Y. Amit, D. Geman, and K. Wilder. Joint induction of shape features and tree classifiers. *IEEE PAMI*, 19(11):1300–1305, 1997.

[3] A. Berg, T. Berg, and J. Malik. Shape matching and object recognition using low distortion correspondence. *CVPR*, 2005.

[4] A. Bosch, A. Zisserman, and X. Muñoz. Scene classification via plsa. *ECCV*, 2006.

[5] A. Bosch, A. Zisserman, and X. Muñoz. Representing shape with a spatial pyramid kernel. *CIVR*, 2007.

[6] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[7] O. Chum and A. Zisserman. An exemplar model for learning object classes. *CVPR*, 2007.

[8] G. Csurka, C. Bray, C. Dance, and L. Fan. Visual categorization with bags of keypoints. In *Workshop on Statistical Learning in Computer Vision, ECCV*, pages 1–22, 2004.

[9] N. Dalal and B. Triggs. Histogram of oriented gradients for human detection. *CVPR*, 2005.

[10] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *IEEE CVPR Workshop of Generative Model Based Vision*, 2004.

[11] A. Frome, Y. Singer, and J. Malik. Image retrieval and classification using local distance functions. *NIPS*, 2006.

[12] K. Grauman and T. Darrell. The pyramid match kernel: Discriminative classification with sets of image features. *ICCV*, 2005.

[13] G. Griffin, A. Holub, and P. Perona. Caltech 256 object category dataset. Technical Report UCB/CSD-04-1366, California Institute of Technology, 2007.

[14] A. Hegerath, T. Deselaers, and H. Ney. Patch-based object recognition using discriminatively trained gaussian mixtures. *BMVC.*, 2006.

[15] I. Laptev. Improvements of object detection using boosted histograms. *BMVC.*, 2006.

[16] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. *CVPR*, 2006.

[17] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *IEEE PAMI*, 2006.

[18] Y. Lin, T. Liu, and C. Fuh. Local ensemble kernel learning for object category recognition. *CVPR*, 2007.

[19] D. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.

[20] F. Moosmann, B. Triggs, and F. Jurie. Fast discriminative visual codebooks using randomized clustering forests. *NIPS*, 2006.

[21] M. Ozuysal, P. Fua, and V. Lepetit. Fast keypoint recognition in ten lines of code. *CVPR*, 2007.

[22] V. Perronnin, C. Dance, G. Csurka, and M. Bressan. Adapted vocabularies for generic visual categorization. *ECCV*, 2006.

[23] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. *ICCV*, 2003.

[24] J. Winn and A. Criminisi. Object class recognition at a glance. *CVPR*, 2006.

[25] J. Winn and J. Shotton. The layout consistent random field for recognizing and segmenting partially occluded objects. *CVPR*, 2006.

[26] H. Zhang, A. Berg, M. Maire, and J. Malik. SVM-KNN: Discriminative nearest neighbor classification for visual category recognition. *CVPR*, 2006.

[27] J. Zhang, M. Marszałek, and C. Lazebnik, S. Schmid. Local features and kernels for classification of texture and object categories: a comprehensive study. *IJCV*, 2007.