Chapter 9
Keypoint Recognition using Random Forests and Random Ferns *

V. Lepetit and P. Fua

Abstract In many 3-D object-detection and pose-estimation problems, run-time performance is of critical importance. However, there usually is time to train the system. We introduce an approach that takes advantage of this fact by formulating wide-baseline matching of keypoints extracted from the input images to those found in the model images as a classification problem. This shifts much of the computational burden to a training phase and eliminates the need for expensive patch preprocessing, without sacrificing recognition performance. This makes our approach highly suitable for real-time operations on low-powered devices.
To this end, we developed two related methods. The first uses Random Forests that rely on simple binary tests on image intensities surrounding the keypoints. In the second, we flatten the trees to turn them into simple bit strings, which we will refer to as Ferns, and combine their output in a Naive Bayesian manner. Surprisingly, the Ferns, while simpler, actually perform better than the trees. This is because the Naive Bayesian approach benefits more from the thousands of synthetic training examples we can generate than output averaging as usually performed by Random Forests. Furthermore, the more general partition the trees allow does not appear to be of great use for our problem.

9.1 Introduction

In many 3–D object-detection and pose estimation problems ranging from Augmented Reality to Visual Servoing, run-time performance is of critical importance. However, there usually is time to train the system before actually using it. Furthermore 3–D models, or multiple images from which such models can be built, tend to

V. Lepetit, P. Fua
Ecole Polytechnique Fédérale de Lausanne, Lausanne (Switzerland)

* This work was supported in part by the Swiss National Science Foundation

be available. As shown in Figure 9.1, we describe here a technique designed to operate effectively in this context by shifting much of the computational burden to the training phase so that run-time detection becomes both fast and reliable. Our general approach, like many others, relies on matching interest points extracted from training images and those extracted from input images acquired at run-time under potentially large perspective and scale variations. It turns out to be very simple to implement, and to perform as well as SIFT [18] while being faster.

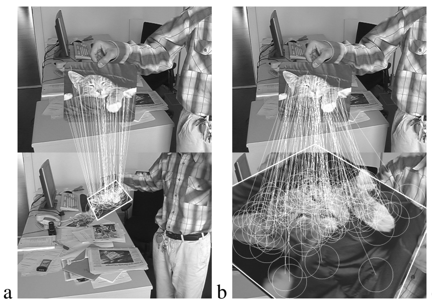
Interest points are usually matched by building affine-invariant descriptors of the surrounding image patches and to compare them across images. This typically involves fine scale selection, rotation correction, and intensity normalization [20, 18]. It results in a high computational overhead and often requires handcrafting the descriptors to achieve insensitivity to specific kinds of distortion.

Instead, we turn this problem into a classification one. More specifically, we consider the set of all possible appearances of each individual object keypoint as a class, which we refer to as a *view set*. During training, given at least one image of the target object, we extract interest points and generate numerous synthetic views of their possible appearance under perspective distortion, which are then used to train a classifier. It is used at run-time to recognize the keypoints under perspective and scale variations by deciding to which view set, if any, their appearance belongs.

We first consider using Classification Forests [2], as described in Chapter **??**, as the classification technique, because they naturally handle multi-class problems. Furthermore, they are robust and fast, while remaining reasonably easy to train. We then show that the trees can be profitably replaced by non-hierarchical structures that we refer to as *Ferns* to classify the patches. Each one consists of a small set of binary tests and returns the probability that a patch belongs to any one of the classes that have been learned during training. These responses are then combined in a Naive Bayesian way. As before, we train our classifier by synthesizing many views of the keypoints extracted from a training image as they would appear under different perspective or scale. Thanks to the Naive Baysian approach, the Ferns are more reliable than the trees, while being much faster and simpler to implement. They do not require *ad hoc* patch normalization, and allow for fast and incremental training.

9.2 Wide Baseline Point Matching as a Classification Problem

Our approach to object detection and pose estimation relies on matching keypoints found in an input image against those on a target object $\mathcal{O}$. Once potential correspondences have been established, we apply standard techniques to estimate the 3–D pose. Therefore, the critical step in achieving results such as those depicted in Figure 9.1 is the fast and robust wide-baseline matching that handling large perspective and scale changes implies, which we formulate below in terms of a classification problem.



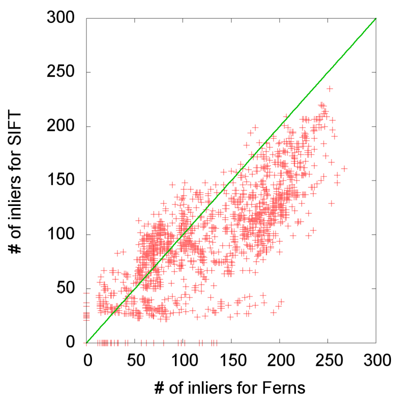


Fig. 9.1 Matching a mouse pad in a 1074-frame sequence against a reference image. (a, b) Matches obtained using ferns in a few frames. The reference image appears at the top and the input image from the video sequence at the bottom. (c) Scatter plot showing the number of inliers for each frame. The values on the x- and y- axes give the number of inliers for the Ferns and SIFT, respectively. Most of the time, the Ferns match at least as many points as SIFT and often even more, as can be seen from the fact that most of the points lay below the diagonal.

During training, we first select a set $\mathcal{K}$ of K prominent keypoints lying on the object model. At runtime, given an input patch $\mathbf{v}$ centered at a keypoint extracted from the input image, we want to decide whether or not its appearance matches that one of the K keypoints in $\mathcal{K}$. In other words, we want to find for $\mathbf{v}$ its class label $c(\mathbf{v}) \in \mathcal{C} = \{-1, 1, 2, \ldots, K\}$, where the -1 label denotes all the points that do not belong to $\mathcal{K}$.

In other tasks such as face detection or character recognition, large training sets of labeled data are usually available. However, for automated pose estimation, it would be impractical to require a very large number of sample images. Instead, to achieve robustness with respect to pose and complex illumination changes, we use a small number of reference images and synthesize many new views of the object using simple rendering techniques. For each keypoint, this gives us a sampling of its view set, the set of all its possible appearances under different viewing conditions. These samplings are virtually infinite training sets. Figure 9.2 depicts such a sampling for several keypoints.

9.3 Keypoint Recognition with Classification Forests

Several classification algorithms, such as K-Nearest Neighbor, Support Vector Machines or neural networks could have been chosen to implement the classifier Y introduced in Section 9.2. Among those, we have found Random Forests, also re-

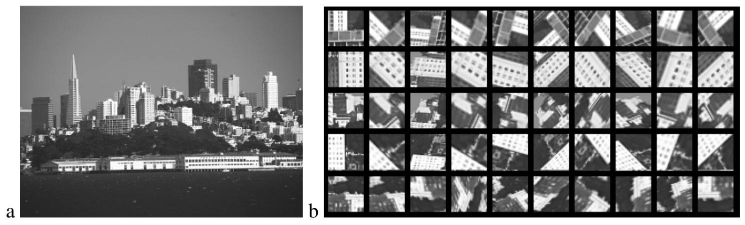


Fig. 9.2 (a) One of our reference images used in the evaluations. (b) Warped patches obtained by applying affine deformations to this image. In each line, the left most patch is the original one and the others are deformed versions of it. They are used to train our algorithms after noise addition.

ferred to as Randomized Trees [2] or Classification Forests, to be eminently suitable because they naturally handle multi-class problems and are robust and fast, while remaining reasonably easy to train. We describe in this section their application to our specific problem. In the next section, we will show how they can be further simplified into another classifier we call Ferns, while improving the performances.

9.3.1 Random Classification Forests

We briefly recall here how Random Forests can be used for classification. Each internal node of a tree contains a simple test that splits the space of data to be classified, in our case the space of image patches. Each leaf contains an estimate based on training data of the posterior distribution over the classes. A new patch is classified by dropping it down the tree and performing an elementary test at each node that sends it to one side or the other. When it reaches a leaf, it is assigned probabilities of belonging to a class depending on the distribution stored in the leaf. Since the numbers of classes, training examples and possible tests are large in our case, building the optimal tree quickly becomes intractable. Instead, multiple trees are grown so that each tree yields a different partition of the space of image patches.

Once the trees $\mathsf{T}_1, \ldots, \mathsf{T}_T$ are built, their responses are combined during classification to achieve a better recognition rate that a single tree could. More formally, the tree leaves store posterior probabilities $p(c \mid L_t = l(t, \mathbf{v})) = p_t(c \mid \mathbf{v})$, where c is a label in $\mathcal{C}$, t is the index of the tree, and $L_t = l(t, \mathbf{v})$ is the leaf of tree T_t reached by patch $\mathbf{v}$. Such probabilities are evaluted during training as the ratio of the number of patches of class c in the training set that reach l and the total number of patches that reach l. Patch $\mathbf{v}$ is classified by considering the average of the probabilities $p(c \mid L_t = l(t, \mathbf{v}))$:

$$\hat{Y}(\mathbf{v}) = \underset{c}{\operatorname{argmax}} \sum_{t=1 \ldots T} p(c \mid L_t = l(t, \mathbf{v})) \,. \tag{9.1}$$

The drawback of classification forests is their greedy use of memory. Their size in memory increases exponentially with the depth, and linearly with the number of trees. For example, a single tree of depth 15 uses about 32 Mb for a 200 classes problem. Therefore, the chosen number of trees and their depth are a trade-off between the computer memory dedicated to store them and the recognition rate. In Section 9.5, we study the influence of these parameters on the recognition rate.

9.3.2 Node Tests

In our implementation, the tests performed at the nodes are simple binary tests based on the difference of intensities of two pixels $\mathbf{p}_1$ and $\mathbf{p}_2$ taken in the neighborhood of the keypoint. We write these tests as

$$h_i(\mathbf{v},(\mathbf{p}_1^i,\mathbf{p}_2^i)) = [\mathcal{J}(\mathbf{v},\mathbf{p}_1^i) \leq \mathcal{J}(\mathbf{v},\mathbf{p}_2^i)]$$

where $\mathcal{J}(\mathbf{v},\mathbf{p})$ is the intensity of patch $\mathbf{v}$ at pixel location $\mathbf{p}$, after Gaussian smoothing to reduce influence of noise. Such a test can be seen as a test on the polarity between the two locations $\mathbf{p}_1^i$ and $\mathbf{p}_2^i$. In all our experiments, the patches are of size 32×32, so that the total number of possible h tests is 2^{19}. Fortunately, since real-world images exhibit spatial coherence, only a very small subset is required to yield good recognition rates.

As shown below, a few hundreds of these simple tests are usually enough to classify a patch. This involves only a few hundreds intensity comparisons and additions per patch, and is therefore very fast. Furthermore, because they only depend on the order of the pixel intensities between neighbors, they tend to be fairly insensitive to illumination changes other than those caused by a moving shadow. In other words, to achieve the robustness to illumination effects demonstrated in Figure 9.1, our technique, unlike many others, does *not* require us to normalize the pixel intensities, for example by setting the L_2 norm of the intensities to one.

9.3.3 Building the Trees

To improve the recognition rate, we use multiple trees that should partition the patches space in different manners. We experimented with two different methods for building such trees.

The first method is the one described in Chapter **??**: The trees are constructed in the classical, top-down manner, where the tests are chosen by a greedy algorithm to best separate the given examples. The expected gain in information is used to evaluate the separation efficiency.

The second method is much faster and simpler: Instead of picking questions according to a criterion, we simply pick a random set, as also done in the Extremely

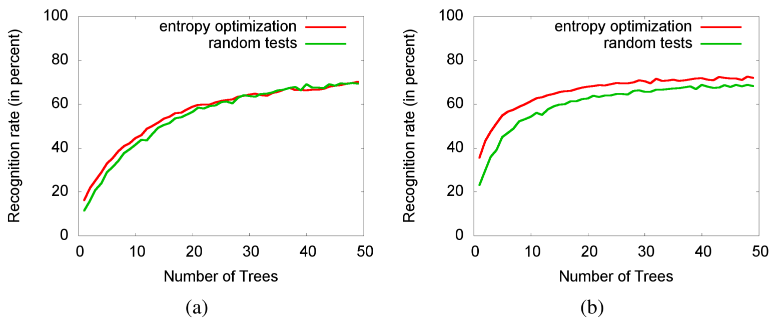


Fig. 9.3 Comparing the classification rates obtained using trees grown in two different manners, as a function of the number of trees. (a) Without and (b) with patch orientation normalization. The thick lines depict results obtained by selecting tests that maximize the information gain. The thin lines depict results obtained by randomly chosen tests, which result in a small loss of reliability but considerably reduces the training time. Note that in all cases the normalization lets us achieve better results with fewer trees. However when enough trees are used, it does not improve the rates anymore.

Randomized Trees [10] approach discussed in the next chapter. This can be seen as an extreme simplification of the first method. The two locations $\mathbf{p}_1^i$ and $\mathbf{p}_2^i$ for each node are picked at random within the patch, independently of the training samples that fall into the node and of the tests performed further up in the tree.

To compare the two tree-building methods we have introduced, we used them both on a set of 200 keypoints. This resulted in two sets of trees whose depth was limited to the same value.

When using the entropy minimizing approach, we first synthesized 100 new views different for each tree to grow. We then recursively built the trees by trying n different tests at each node and keeping the best one according to the information gain. For the root node, we chose $n = 10$, a very small number, to reduce the correlation between the resulting trees. For all other nodes, we used $n = 100d$, where d is the depth of the node. Note that this heuristic involves randomizing on both tests and training data. We do the latter mostly to make our greedy algorithm tractable.

In the case of the completely random approach to building trees, $\mathbf{p}_1^i$ and $\mathbf{p}_2^i$ were simply chosen at random. For the two sets, the tree depth is limited to a given maximal depth, and the posterior probabilities are estimated from 1000 new random views per keypoint.

For this experiment, we used trees with a depth limited to 12, which was found to be a good trade-off between the memory requirements and recognition rate. After having grown the trees, the posterior probabilities in the terminal nodes were estimated using 5000 new training images. We then measured the recognition rate R of the two sets of trees by generating new images under random poses, as the ratio of the number of correctly recognized patches and the total number of generated patches. The evolution of R for the two sets of trees with respect to the number of trees is depicted Figure 9.3(a). Taking the tests at random usually results in a small loss of reliability at least when the number of trees is not large enough but considerably reduces the learning time. The time dedicated to growing the trees drops from tens of minutes to a few seconds on a 2.8 GHz machine.

We also experimented with normalizing the **v** patches' orientations both during training and at run-time to achieve higher recognition rates for a given number of trees. As in [18] we attribute a 2–D orientation to the keypoints that is estimated from the histogram of gradient directions in a patch centered at the keypoint. Note that by contrast with [18], we do not require a particularly repeatable method. We just want it to be reliable enough to reduce variation within classes. Once the orientation of an extracted keypoint is estimated, its neighborhood is rectified. Figure 9.3(b) compares the recognition rates with this normalization step for the two different methods of selecting the tests. Taking the tests at random results in a slightly larger but still small loss of reliability. More importantly, the normalization gives us significantly improved rates when using only a small number of trees. However, when using a large number of trees, the recognition rates are similar with and without the normalization.

We draw two practical conclusions from these experiments. First, using random tests is sufficient and keeps the learning time reasonable for practical applications. Second, the orientation normalization step is not required, but lets us reduce the number of trees. Therefore the choice of using such a normalization becomes a trade-off between the amount of time required to normalize and to classify, which is proportional to the number of trees. However, in the next section, we discuss an approach closely related to trees that reaches, without normalization, performances similar to the trees when normalization is used, for an equal amount of computations.

9.4 Keypoint Recognition with Random Ferns

In this section, we will argue that, when the tests are chosen randomly, the power of our general approach derives not from the tree structure itself but from the fact that combining groups of binary tests yields improved classification rates. To this end, we drop the hierarchical structure of the trees and group the tests into a flat one we call Fern. We first show that our Ferns fit nicely into a Naive Bayesian framework and yield better results and scalability in terms of number of classes. As a result, we can combine many more features, which is key to increasing performance.

9.4.1 Random Ferns

Our general approach is still similar to the one taken with the Randomized Trees in the previous section: Given the patch surrounding a keypoint detected in an image, our task is to assign it to the most likely class. Let $h_j = h_j(\mathbf{v}, (\mathbf{p}_1^j, \mathbf{p}_2^j)), j = 1, \ldots, N$ be the set of binary features that will be calculated over the patch $\mathbf{v}$ we are trying to classify. Formally, we are looking for

$$\hat{Y}(\mathbf{v}) = \underset{c}{\operatorname{argmax}}\, p(c \mid h_1, h_2, \ldots, h_N) \,,$$

where C is a random variable that represents the class. Bayes' Formula yields

$$p(c \mid h_1, h_2, \ldots, h_N) = \frac{p(h_1, h_2, \ldots, h_N \mid c) p(c)}{p(h_1, h_2, \ldots, h_N)} \,.$$

Assuming a uniform prior $p(C)$, since the denominator is simply a scaling factor that it is independent from the class, our problem reduces to finding

$$\underset{c}{\operatorname{argmax}}\, p(h_1, h_2, \ldots, h_N \mid c) \,. \tag{9.2}$$

Since the h_j features are very simple, we require many ($N \approx 300$) for accurate classification. Therefore a complete representation of the joint probability in Eq. (9.2) is not feasible since it would require estimating and storing 2^N entries for each class. One way to compress the representation is to assume independence between features. An extreme version is to assume complete independence, that is,

$$p(h_1, h_2, \ldots, h_N \mid c) = \prod_{j=1}^{N} p(h_j \mid c) \,.$$

However this completely ignores the correlation between features. To make the problem tractable while accounting for these dependencies, a good compromise is to partition our features into F groups of size $S = \frac{N}{F}$. These groups are what we define as *Ferns* and we compute the joint probability for features in each Fern. The conditional probability becomes

$$p(h_1, h_2, \ldots, h_N \mid c) = \prod_{f=1}^{F} p(\mathsf{F}_k \mid c) \,, \tag{9.3}$$

where $\mathsf{F}_f = \{h_{\sigma(f,1)}, h_{\sigma(f,2)}, \ldots, h_{\sigma(f,S)}\}, f = 1, \ldots, F$ represents the f^{th} fern and $\sigma(f, j)$ is a random permutation function with range $1, \ldots, N$. Hence, we follow a Semi-Naive Bayesian [30] approach by modelling only some of the dependencies between features. The viability of such an approach has been shown by [16] in the context of image retrieval applications. In this new method, patch $\mathbf{v}$ is therefore classified using:

$$\hat{Y}(\mathbf{v}) = \underset{c}{\operatorname{argmax}} \prod_{f=1}^{F} p(\mathsf{F}_f \mid c) \,. \tag{9.4}$$

This formulation yields a tractable problem that involves $F \times 2^S$ parameters, with F between 30-50. In practice, as will be shown in Section 9.5, $S = 11$ yields good results. $F \times 2^S$ is therefore in the order of 80,000, which is much smaller than 2^N with $N \approx 450$ that the full joint probability representation would require. Our formulation is also flexible since performance/memory trade-offs can be made by changing the number of Ferns and their sizes.

Note that we use randomization in feature selection but also in grouping. An alternative approach would involve selecting feature groups to be as independent from each other as possible. This is routinely done by Semi-Naive Bayesian classifiers based on a criteria such as the mutual information between features. However, in practice, we have not found this to be necessary to achieve good performance. We have therefore chosen not to use such a strategy to preserve the simplicity and efficiency of our training scheme and to allow for incremental training.

9.4.2 Training the Ferns

The training phase estimates the class conditional probabilities $p(\mathsf{F}_f \mid c)$ for each Fern F_f and class c, as described in Eq. (9.3). For each Fern F_f we write these terms as:

$$p_{k,c} = p(\mathsf{F}_f = k \mid c)\,, \tag{9.5}$$

where we simplify our notations by considering F_f to be equal to k if the base 2 number formed by the binary features of F_f taken in sequence is equal to k. With this convention, each Fern can take $K = 2^S$ values and we need to estimate the $p_{k,c}, k = 1,2,\ldots,K$ under the constraint that their sums over k should be equal to 1. The simplest approach would be to assign the maximum likelihood estimate to these parameters from the training samples. For parameter $p_{k,c}$ it is

$$p_{k,c} = \frac{N_{k,c}}{N_c}\,,$$

where $N_{k,c}$ is the number of training samples of class c that evaluates to Fern value k and N_c is the total number of samples for class c. These parameters can therefore be estimated for each Fern independently.

In practice however, this simple scheme yields poor results because if no training sample for class c evaluates to k, which can easily happen when the number of samples is not infinitely large, both $N_{k,c}$ and $p_{k,c}$ will be zero. Since we multiply the $p_{k,c}$ for all Ferns, it implies that, if the Fern evaluates to k, the corresponding patch can *never* be associated to class c, no matter the response of the other Ferns. This would make the Ferns far too selective because the fact that $p_{k,c} = 0$ may simply be an artifact of the necessarily limited size of the training set. To overcome this problem we take $p_{k,c}$ to be

$$p_{k,c} = \frac{N_{k,c} + N_r}{N_c + K \times N_r},$$

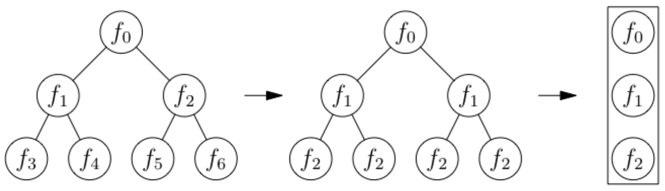


Fig. 9.4 Ferns vs Trees. A tree can be transformed into a Fern by performing the following steps. First, we constrain the tree to systematically perform the same test across any given hierarchy level, which results in the same feature being evaluated independently of the path taken to get to a particular node. Second, we do away with the hiearchical structure and simply store the feature values at each level. This means applying a sequence of tests to the patch, which is what Ferns do.

where N_r represents a regularization term, which behaves as a uniform Dirichlet prior [4] over feature values. If a sample with a specific Fern value is not encountered during training, this scheme will still assign a non-zero value to the corresponding probability. We have found our estimator to be insensitive to the exact value of N_r and we use $N_r = 1$ in all our experiments. However, having N_r be strictly greater than zero is essential. This tallies with the observation that combining classifiers in a Naive Bayesian fashion can be unreliable if improperly done.

In effect, our training scheme marginalizes over the pose space since the class conditional probabilities $P(\mathsf{F}_f \mid c)$ depend on the camera poses relative to the object. By densely sampling the pose space and summing over all samples, we marginalize over these pose parameters. Hence at run-time, the statistics can be used in a pose independent manner, which is key to real-time performance. Furthermore, the training algorithm itself is very efficient since it only requires storing the $N_{k,c}$ counts for each fern while discarding the training samples immediately after use, which means that we can use arbitrarily many if need be.

9.5 Comparing Random Forests, Random Ferns, and SIFT

9.5.1 Empirical Comparisons of Trees and Ferns

Ferns differ from trees in two important respects: As shown in Figure 9.4, Ferns can be considered as simplified trees. Also, as can be easily seen by comparing Eqs. (9.1) and (9.4), the trees average posteriors while the ferns rely on products of conditional probabilities. Whether or not the differences degrade the classification performance hinges on whether our randomly chosen binary features are still appropriate in this context. In this section, we will show that they are indeed. In fact, because our Naive Bayesian scheme outperforms the averaging of posteriors, the Ferns are both simpler and more powerful.

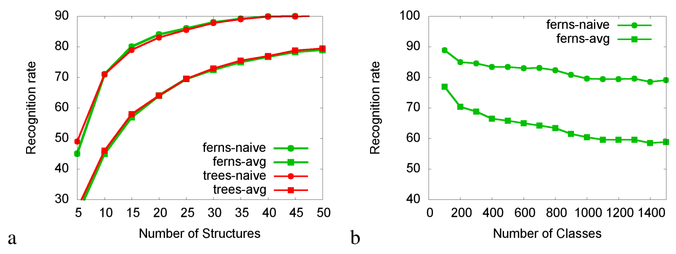


Fig. 9.5 Average percentage of correctly classified image patches over many trials (recognition rate) for Randomized Trees of depth 11 and Random Ferns with 11 features each. (a) Recognition rate as a function of the the number of Trees or Ferns. Using the Naive Bayesian assumption gives much better rates at reduced number of structures, while the Fern and tree structures are interchangeable. (b) Recognition rate as a function of the number of classes. While the naive combination produces a very slow decrease in performance, posterior averaging exhibits a much sharper drop.

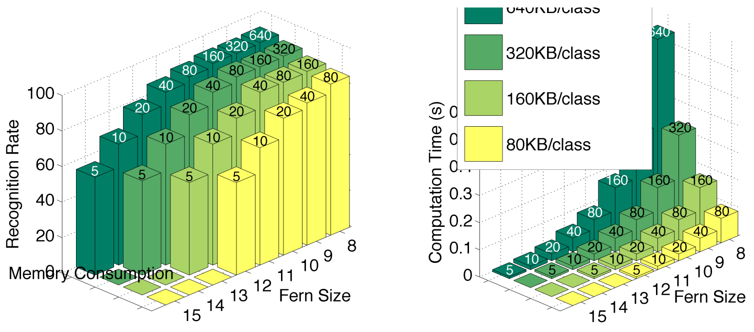
To compare RTs and Ferns, we experiment with three reference images including the one shown in Figure 9.2. We extracted stable keypoints from these images and assigned a unique class id to each of them. The classification is done using patches that are 32×32 pixels in size. To disentangle the influence of the differences between trees and ferns, we consider four different scenarios:

- Using Randomized Trees and averaging of class posterior distributions,
- Using Randomized Trees and combining class conditional distributions in a Naive-Bayesian way,
- Using Ferns and averaging of class posterior distributions,
- Using Ferns and combining class conditional distributions in a Naive-Bayesian way.

Also the number of features evaluated per patch by the two classifiers is equal in all cases. As explained in Section 9.2, the training and testing sets are obtained from the reference images. We randomly deform these images with affine deformations that can arbitrarily rotate the images, skew and scale them over a large range, and add Gaussian noise. More details on this experimental setup can be found in [21].

In Figure 9.5a, we plot the results as a function of the number of trees or Ferns being used. We first note that using either flat Fern or hierarchical tree structures does not affect the recognition rate, which was to be expected as the features are taken completely at random. By contrast the Naive-Bayesian combination strategy outperforms the averaging of posteriors and achieves a higher recognition rate even when using relatively few structures.

Figure 9.5b shows that the performance of the Naive-Bayesian combination does not degrade rapidly with the number of classes and scales much better than averaging posteriors. For both methods, the required amounts of memory and computation times increase linearly with the number of classes, since we assign a separate class for each keypoint.



(a) (b)

Fig. 9.6 Recognition rate (a) and computation time in seconds (b) as a function of the amount of memory available and the size of the Ferns being used. The number of Ferns used is indicated on the top of each bar and the y-axis shows the Fern size. The color of the bar represents the required memory amount, when using single precision floating numbers. Note that while using many small ferns achieves higher recognition rates, it also entails a higher computational cost.

Increasing the Fern size by one doubles the number of parameters hence the memory required to store the distributions. It also implies that more training samples should be used to estimate the increased number of parameters. It has however negligible effect on the run-time speed and larger Ferns can therefore handle more variation at the cost of training time and memory but without much of a slow-down.

By contrast adding more Ferns to the classifier requires only a linear increase in memory but also in computation time. Since the training samples for other Ferns can be reused it only has a negligible effect on training time. As shown in Figure 9.6, for a given amount of memory the best recognition rate is obtained by using many relatively small Ferns. However this comes at the expense of run-time speed and when sufficient memory is available, a Fern size of 11 represents a good compromise, which is why we have used this value in the experiments.

9.5.2 Empirical Comparisons between SIFT and Ferns

We used the 1074-frame video depicted by Figure 9.1 to compare Ferns against SIFT for planar object detection. It shows a mouse pad undergoing motions involving a large range of rotations, scalings, and perspective deformations against a cluttered background. The graph on the right shows that the Ferns can match as many points as SIFT and sometimes even more.

It is difficult to perform a completely fair speed comparison between our Ferns and SIFT for several reasons. SIFT reuses intermediate data from the keypoint extraction to compute canonical scale and orientations and the descriptors, while ferns can rely on a low-cost keypoint extraction. On the other hand, the distributed SIFT

C code is not optimized, and the Best-Bin-First KD-tree of [3] is not used to speed up the nearest-neighbor search. However, it is relatively easy to see that performing the individual tests of Section 9.3.2 requires very little time and most of the time is spent computing the sums of the posterior probabilities. Computing the SIFT descriptors, which is the most difficult part to optimize, takes about 1ms on a MacBook Pro laptop without including the time required to convolve the image. By contrast, Ferns take 13.5 10^{-3} milliseconds to classify one keypoint into 200 classes on the same machine. Of course, the ability to classify keypoints fast comes at the cost of requiring a training stage, which is usually off-line. By contrast, SIFT does not require training and for some applications such as matching of arbitrary images, this is still clearly an advantage.

9.6 Discussion

The key conclusion of our work is that, in our specific context, the Naive-Bayesian combination of classifiers as done by the Ferns clearly outperforms the averaging of probabilities, as in the case of Random Forests. While we do not know of a clear theoretical argument explaining the superiority of Naive-Bayesian techniques for our purposes, there are pragmatic reasons for choosing them. First, the product models can represent much sharper distributions [13] (see also Chapter 1). Indeed, when averaging is used to combine distributions, the resulting mixture has higher variance than the individual components. More intuitively, if a single Fern strongly rejects a keypoint class, it can counter the combined effect of all the other Ferns that gives a weak positive response. This increases the necessity of larger amounts of training data and the help of a prior regularization term as discussed in Section 9.2. Second, the classification task, which just picks a single class, will not be adversely affected by the approximation errors in the joint distribution as long as the maximum probability is assigned to the correct class [9, 8]. We have shown that such a naive combination strategy is a worthwhile alternative when the specific problem is not overly sensitive to the implied independence assumptions.

9.7 Application Example

We present in this section an application of our approach to real-time image annotation. With the recent proliferation of ultra mobile platforms with higher processing power, there has been a surge of interest in building real-world applications that can automatically annotate the photos and provide useful information about places of interest. These applications test keypoint matching algorithms to their limits under constantly changing lighting conditions and with changes in the scene texture that reduces the amount of reliable keypoints. We have tested the Ferns on such an

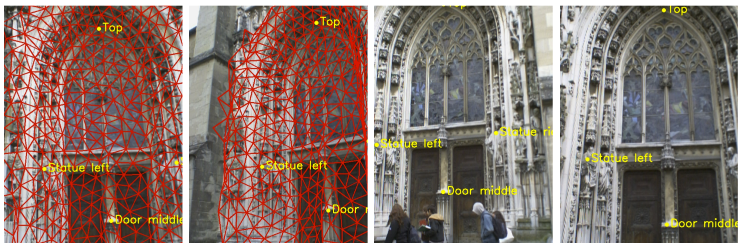


Fig. 9.7 Annotation of a cathedral door using a 3–D model. The first two images also show the 3–D model that is used to estimate the camera position which allows us to reproject the annotation correctly.

application that annotates parts of a historical building with 3–D structure. It runs smoothly at frame rate using a standard laptop and an of the shelf web camera.

Annotating a 3–D object requires training using multiple images from different viewpoints. Thanks to the Ferns approach, we can easily integrate the information from several images. We then obtain a 3–D model for the object using standard structure from motion algorithms to register the training images followed by dense reconstruction [26]. The resulting fine mesh was too detailed and we approximated it by a coarser one. Despite its rough structure, this 3–D model allows annotation of important parts of the object and the correct reprojection of this information onto the image plane under change in viewpoint as depicted by Figure 9.7.

9.8 Conclusion

We have presented a simple yet powerful approach for image patch recognition that performs well even in the presence of severe perspective distortion. The Ferns prove to be particularly adapted, as their “semi-naive” approach yields a scalable, simple, fast, and powerful implementation.

References

1. Y. Amit and D. Geman. Randomized inquiries about shape; an application to handwritten digit recognition. Technical Report 401, Dept. of Statistics, University of Chicago, IL, Nov 1994.
2. Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural Computation*, 9(7):1545–1588, 1997.
3. J. Beis and D.G. Lowe. Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 1000–1006, 1997.
4. C.M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., 2006.
5. L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
6. L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and Regression Trees*. Chapman and Hall/CRC, 1984.
7. A. Criminisi, J. Shotton, D. Robertson, and E. Konukoglu. Regression forests for efficient anatomy detection and localization in CT studies. In *MICCAI workshop on Medical Computer Vision: Recognition Techniques and Applications in Medical Imaging*, Beijing, 2010. Springer.
8. P. Domingos, M. Pazzani, and G. Provan. On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. In *Machine Learning*, pages 103–130, 1997.
9. J. H. Friedman and U. Fayyad. On Bias, Variance, 0/1-loss, and the Curse-of-Dimensionality. *Data Mining and Knowledge Discovery*, 1:55–77, 1997.
10. P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006.
11. T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.
12. D. Heath, S. Kasif, and S. Salzberg. Induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2(2):1–32, 1993.
13. G. E. Hinton. Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation*, 14:1771–1800, 2002.
14. T. K. Ho. Random decision forests. In *Proc. Intl. Conf. on Document Analysis and Recognition*, pages 278–282, 1995.
15. T. K. Ho. The random subspace method for constructing decision forests. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.
16. D. Hoiem, R. Sukthankar, H. Schneiderman, and L. Huston. Object-Based Image Retrieval Using the Statistical Structure of Images. *Journal of Machine Learning Research*, 02:490–497, 2004.
17. Y. Lin and Y. Jeon. Random forests and adaptive nearest neighbors. *Journal of the American Statistical Association*, 2002.
18. D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. Journal on Computer Vision*, 60(2):91–110, November 2004.

19. B. Menze, B.M. Kelm, D.N. Splitthoff, U. Koethe, and F. A. Hamprecht. On oblique random forests. In *Proc. European Conf. on Machine Learning (ECML/PKDD)*, 2011.
20. K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. van Gool. A Comparison of Affine Region Detectors. *Int. Journal on Computer Vision*, 65(1/2):43–72, 2005.
21. M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua. Fast keypoint recognition using random ferns. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 32(3), 201.
22. J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.
23. T. Sharp. Implementing decision trees and forests on a GPU. In *Proc. European Conf. on Computer Vision*, 2008.
24. J. Shotton, A.W. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from a single depth image. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2011.
25. J. Shotton, M. Johnson, and R. Cipolla. Semantic texton forests for image categorization and segmentation. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 1–8, 2008.
26. C. Strecha, R. Fransens, and L. van Gool. Combined Depth and Outlier Estimation in Multi-View Stereo. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2006.
27. Z. Tu. Probabilistic boosting-tree: Learning discriminative models for classification, recognition, and clustering. In *Proc. Intl. Conf. on Computer Vision*, volume 2, pages 1589–1596, Beijing, China, October 2005.
28. P. Viola and M. J. Jones. Robust real-time face detection. *Int. Journal on Computer Vision*, 2004.
29. P. Yin, A. Criminisi, J. Winn, and I. Essa. Tree based classifiers for bilayer video segmentation. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2007.
30. F. Zheng and G.I. Webb. A Comparative Study of Semi-Naive Bayes Methods in Classification Learning. In *Australasian Data Mining Conference*, pages 141–156, 2005.