

Improving ferns ensembles by sparsifying and quantising posterior probabilities

Antonio L. Rodriguez

Vitor Sequeira

Joint Research Centre, Institute for Transuranium Elements. Ispra, Italy.

`{antonio.rodriguez,vitor.sequeira}@jrc.ec.europa.eu`

Abstract

Ferns ensembles offer an accurate and efficient multi-class non-linear classification, commonly at the expense of consuming a large amount of memory. We introduce a two-fold contribution that produces large reductions in their memory consumption. First, an efficient L_0 regularised cost optimisation finds a sparse representation of the posterior probabilities in the ensemble by discarding elements with zero contribution to valid responses in the training samples. As a by-product this can produce a prediction accuracy gain that, if required, can be traded for further reductions in memory size and prediction time. Secondly, posterior probabilities are quantised and stored in a memory-friendly sparse data structure. We reported a minimum of 75% memory reduction for different types of classification problems using generative and discriminative ferns ensembles, without increasing prediction time or classification error. For image patch recognition our proposal produced a 90% memory reduction, and improved in several percentage points the prediction accuracy.

1. Introduction

Initially proposed in [1], random forest classifiers offered interesting computational advantages over previous classification techniques. A random forest basically consist of an ensemble of random decision trees. Given an input item that should be classified, each random tree emits a prediction for each one of the different possible classes. The random forest averages these predictions to aggregate them into a single, better classification.

Several contributions improved the performance and reduced the computational requirements of these classifiers, making them more suitable for a wider range of applications. In [20] the authors introduced ferns ensembles. Unlike random trees, ferns evaluate a fixed set of binary features on the input sample. This made unnecessary the large tree structures of binary tests, and made more efficient the

processing of each new input sample. Ferns ensembles also replaced the simple addition of class probabilities, used to combine trees predictions in random forests, with the naive Bayes aggregation, improving significantly the classification accuracy for tasks such as image feature recognition [19, 15]. Ferns ensembles are commonly trained using a generative formulation. Given an input sample the classifier estimates the probability for each class. The output prediction will be the class with the highest probability. Ferns can also be trained using a discriminative approach where misclassifications are directly penalised. For example by minimising a cost similar to the regularised Hinge loss, as it is done for the training of support vector machines (SVM). A ferns ensemble could be trained by optimising the multiclass version of the margin error [5, 10]. In [13] the authors optimise the regularised Hinge loss producing one-vs-all predictions for each single class, that are later averaged to produce the aggregated prediction.

Their inherent non-linear multiclass formulation, their efficiency, and accuracy made random forests and ferns ensembles specifically suited for real-time applications where a fast, accurate classification is needed. These classifiers achieved a significant popularity for visual recognition tasks such as image patch classification [14] and human pose estimation [24]. Once trained, evaluating the probabilities for a thousand classes will usually take a few milliseconds using commodity hardware. A small number of pixel-wise comparisons of values in regular or depth images, and a small number of additions, are enough to recognise image patches or features with high accuracy. These classifiers have also been used for a large variety of problems such as object categorisation [27], image classification [4], simultaneous location and mapping [6, 19], unsupervised detection and tracking [22], sensor relocalisation [11], face recognition [8], vehicle detection and tracking [9], pedestrian detection [18], or hand pose estimation [13].

Most of these proposals provided state-of-the-art results for the specific problem. However, random forests and ferns usually trade their computational advantages for the disadvantage of consuming a large amount of memory. The

memory consumption of forests and fern classifiers is dominated by the size of the posterior probabilities stored in memory, which can become excessive when the number of classes is large, or when the classifier must use a high number of ferns or binary tests to achieve a sufficient accuracy. Furthermore, the classifier can easily deplete the CPU memory when using low-end hardware, or when the application requires using large number of classifiers.

In this paper we exploit two different facts to obtain large reductions in the memory consumption of ferns ensembles. We demonstrate that ferns typically use a large number of posterior probabilities that are non-essential to produce accurate classifications. Furthermore, we also demonstrate that a reasonable quantisation of the probabilities will usually have a neglectable effect in the prediction accuracy. Exploiting these facts we developed a method to post-process the probabilities after the training that optimises their memory size. It is composed of the following parts, that also summarize the main paper contributions:

- A method to quantise probabilities, reducing the memory space that they individually occupy.
- A highly efficient L_0 regularised optimisation that discards elements in the ferns with a reduced contribution to the classification margin error. Thanks to an adequate linear transformation we can set to zero those elements without degrading the prediction accuracy.
- A memory-friendly sparse container that maintains the remaining elements in memory, so they can be used by the classifier to emit predictions.

The optimisation is a selection procedure based on TISP[23]. It can also be considered as a relaxation of the multiclass margin error minimisation [10]. The L_0 regularisation included in the proposed loss function ensures high sparsity in the solution obtained. Commonly, costs that include L_0 regularisations tend to be difficult to optimise. They use to be non-convex and have a large number of local minima. However, the proposed optimisation is separable, thanks to its simplicity. Each component of the solution can be found using an efficient closed-form method. The optimisation can compress the memory size of ferns ensembles that were trained using either generative or discriminative procedures. In the latter case, due to the L_2 regularisation introduced in the Hinge loss function, ferns can have a certain degree of sparsity. However, as shown in the results section our optimisation still produces large increases in the sparsity without degrading prediction accuracy.

We evaluated the advantages and performance gain provided by the post-processing with generative and discriminative ferns ensembles, performing tests for different kinds of classification problems, and comparing the results obtained with and without the compression. In these tests

we used publicly available model images and data-sets that have been used previously in computer vision and machine learning literature. For image patch recognition, our method reduced the memory size in one order of magnitude. Meanwhile, it increased between 3 and 5 percentage points the prediction accuracy. In this case the classification time remained similar. The post-processing can also balance memory consumption, speed and accuracy, to provide, for example, a classifier with a prediction accuracy equivalent to the one obtained with the original ferns ensemble. In this case the post-processing can decrease the prediction time and obtain larger memory reductions. Sparse containers occupy a smaller amount of memory than regular ones when the number of zero elements is sufficiently high. Due to the extra indexing data that these structures have to maintain, the opposite can be true. If the ferns are not sparse enough, the sparse structure can require more memory than a regular dense container. This was the case for some classification problems in our tests where our proposal did not provide ferns with enough sparsity. In these cases however, the quantisation ensured a minimum memory reduction of 75%, as further discussed below.

1.1. Related work

The advantages of sparse model representations in terms of improved performance, smaller memory footprint, and fast access time have been exploited in different areas such as machine learning [3] or 3D reconstruction [12]. Quantisation techniques have been previously used in SVM to speed up the training [17] and reduce the precision of the classifier parameters in order to adequate them to specific requirements of certain hardware architectures [2, 16]. To the best of our knowledge this is the first proposal that provides large reductions in the memory consumption of ferns ensembles by quantising and obtaining a sparse representation of the probabilities in the ferns for their efficient memory storage in a sparse container.

Different proposals improved and reduced memory consumption of random forests and ferns ensembles. It is possible to optimise and balance memory consumption, speed, and accuracy by adjusting the number of ferns and binary tests per fern [19]. Using more samples in the training can also improve classification time and accuracy [13]. The speed can also be optimized by selecting on demand the number of trees that are consulted for each prediction [21]. In [18] the authors combine random forests with support vector machines (SVM) performing a highly accurate pedestrian detection. Each binary test is evaluated by a SVM trained on a subset of the input features. Since the proposed post-processing method reduces the size of ferns in memory after the training and does not change the prediction algorithm, it can be used in combination with any of these techniques to obtain an improved performance.

Visual recognition problems such as image patch classification have been commonly solved using image feature descriptors. Ferns combine a small prediction time and high accuracy, that is competitive with the fastest binary descriptors[7, 25], with the invariance to rotations and reasonable perspective deformations comparable to that offered by more sophisticated descriptors such as SIFT and SURF[26]. In [6] the authors showed that the response vector of ferns ensembles containing the predicted class probabilities is usually sparse. They exploited this fact to compress the ferns in memory after the training using a random projection sensing matrix. This way the ferns ensemble does not produce a classification, but a reduced image descriptor for each input image patch. However, this descriptor is not invariant to certain transformations such as camera rotations. A similar approach is proposed in [11] where the authors introduced a variant of ferns ensembles for camera relocalisation using RGBD images and on-line training. In this approach the binary tests in the ferns are directly used to produce a binary descriptor for each input key-frame. The posterior probabilities become non-essential for the classification, and storing them in memory is not required at the expense of sacrificing the prediction invariance. Our approach does not discard the posterior probabilities. After the post-processing the predictions of the fern classifier will be highly accurate and invariant to the usual transformations.

1.2. Structure of this document

Section 2 describes the generative and discriminative approaches for training ferns ensembles and introduces the formulation that will be used in the following sections. Section 3 describes the quantisation procedure that reduces the individual memory size occupied by each probability value after the training. Section 4 describes the method used to obtain the sparse representation of the ferns probabilities. Section 5 describes the efficient sparse storage used to take advantage of both the sparsity and the quantisation, to store the probabilities using a reduced amount of memory. Section 6 provides experimental results that endorse the advantages described for the algorithm. The paper concludes with section 7 that reviews the main contributions and conclusions of the paper.

2. Generative and discriminative ensembles

Each tree in a random forest typically evaluates n_t binary tests on a given input item that is presented for its classification, being n_t the depth of the tree. The combination of binary values is used to retrieve a leaf in the tree, which is a vector containing one estimated probability for each class. Most implementations store the leaves of the trees in a table L of size $n_f \times 2^{n_t} \times n_c$, being n_f the number of trees in the forest, and n_c the number of classes predicted. The forest can convert the binary values evaluated by each tree

into a number d and use it to index the leaves in this table. The class c^* for the input item is predicted using the list of indexes $\mathbf{d} = \{d_i\}_{i=1..n_f}$ generated by the trees in the forest from the item as follows:

$$c^* = \arg \max_k \sum_{i=1}^{n_f} L_{i,d_i,k} \quad (1)$$

Provided a training set of sample items labeled with their corresponding valid classes, the training usually generates a table A of size $n_f \times 2^{n_t} \times n_c$ containing the number of times each leaf is associated to each class. With this matrix the elements in L can be estimated as follows:

$$L_{i,j,k} \simeq \frac{A_{i,j,k}}{\sum_{k=1}^{n_c} A_{i,j,k}} \quad (2)$$

Each element $A_{i,j,k}$ contains the number of training samples that are labeled with the k -th class, and simultaneously have the i -th index value set to $d_i = j$.

2.1. Generative ferns training

With the naive Bayes scheme introduced by ferns ensembles the class c^* for a given input is inferred as follows:

$$c^* = \arg \max_{c_k} P(c_k | d_1, d_2, \dots, d_{n_f}) \quad (3)$$

Estimating the probability $P(c_k | d_1, d_2, \dots, d_{n_f})$ for each class and possible combination of values for the indexes is usually intractable. Assuming the statistical independence of the indexes we can rewrite it as follows:

$$P(c_k | d_1, d_2, \dots, d_{n_f}) = \frac{P(c_k) \prod_{i=1}^{n_f} P(d_i | c_k)}{P(d_1, d_2, \dots, d_{n_f})} \quad (4)$$

This approximation works adequately for classification problems such as image patch recognition, and the probabilities $P(d_i | c_k)$ and $P(c_k)$ can be computed from the matrix of frequencies A as follows:

$$P(d_i = j | c_k) \simeq \frac{A_{i,j,k} + \delta}{\sum_{k=1}^{n_c} A_{i,j,k} + \delta}, \quad P(c_k) \simeq \frac{2^{n_t} \sum_{j=1}^{2^{n_t}} A_{i,j,k} + \delta}{\sum_{j=1}^{2^{n_t}} \sum_{i=1}^{n_f} A_{i,j,k} + \delta} \quad (5)$$

Therefore, equation 3 becomes the naive Bayes rule:

$$c^* = \arg \max_{c_k} P(c_k) \prod_{i=1}^{n_f} P(d_i | c_k) \quad (6)$$

The non-zero constant value δ (normally set to $\delta = 1$) can be interpreted as a Dirichlet prior [20]. It should be included to ensure that using equations 5 and 6 does not

produce unstable results in the ferns ensemble predictions due to the multiplicative nature of the Bayes aggregation. To speed-up the classification of new inputs we can map the probabilities to the logarithmic space. The evaluation of the naive Bayes aggregation becomes identical to equation 1 by computing matrix L as follows:

$$L_{i,j,k} = \log P(d_i = j|c_k) + \frac{1}{n_f} \log P(c_k) \quad (7)$$

Commonly, matrix A will be highly sparse, as only few classes in each leaf will be associated to training samples. In random forests the matrices A and L will have zero values at the same elements. Due to the δ increment in equations 5 the expression 7 will not produce zero values.

2.2. Discriminative ferns training

With the discriminative approach proposed in [13] each fern in the ensemble is trained separately. The fern output for each class is provided by a one-vs-all classifier, obtained with a convex regularised Hinge loss optimisation similar to the one used in the training of SVM. Hence the discriminative training solves $n_f \times n_k$ optimisations to produce matrix L . After the training, the response of each fern for a given training sample is close to 1 for the labeled class, and to 0 for the remaining classes. The training estimates the elements in each slice of matrix L for a given number of fern i and class k :

$$\mathbf{l}_{i,k} = (L_{i,1,k}, L_{i,2,k}, \dots, L_{i,2^{n_t},k})$$

by minimising the following functional:

$$\frac{1}{2} \|\mathbf{l}_{i,k}\|^2 + C \sum_{u=1}^{n_s} [1 - y_u \mathbf{l}_{i,k}^T \mathbf{b}_u]_+ \quad (8)$$

where $[x]_+ = \max\{x, 0\}$ is the Hinge loss, and n_s is the number of training samples. The value y_u is 1 if the label for the u -th sample in the training set is equal to k , and zero otherwise. The vector \mathbf{b} of size 2^{n_t} contains zero values, except for the value 1 at the element d_i . Due to the L_2 regularisation in the optimised cost, the discriminative training generates a L matrix with a certain amount of zero values. However, as discussed further the sparsity in these leaves can be increased with the proposed method.

3. Quantisation of values in the leaves

After the generative or discriminative training we scale the values in matrix L with the following transformation:

$$L'_{i,j,k} = 2^q \frac{L_{i,j,k} - \min(L)}{\max(L) - \min(L)} \quad (9)$$

Given that it is linear, it will change the responses of the ferns ensemble but not the results provided by the classification rule in equation 1. In this expression q is a positive

non-zero value that represents the desired bit-depth of the posterior probabilities. In our implementation after the linear transformation we round each value in L' down to the closest integer. This way we can store the values in the leaves using a smaller amount of memory.

Typically the values in the leaves of ferns ensembles are stored using a single-precision floating-point number that occupies four bytes in memory [20, 13]. As we will show in the results section, for a value of $q = 8$ the quantisation round-off has a neglectable effect in the prediction accuracy of the ferns ensemble. This way each element in the leaves can contain up to 256 different values. Hence, we can store each value in the leaves using a single byte in memory. With a regular non-sparse storage this would represent a reduction of 75% in the memory space occupied by the leaves. Next section discusses how to increment the number of zero values in the leaves, so we can get even further memory reductions for certain types classification problems using the sparse storage.

4. L_0 maximisation of valid responses

There will commonly be a few zero values in the matrix L' (in most occasions only one value). The proposed optimization will find a sparser version of this matrix, while preserving weights in L' that are both large and have a positive contribution to the classification success rate. Instead of reducing the number of misclassified samples in the training set, as it is done in multiclass margin error optimisation, our procedure aims to set to zero the largest amount of elements in the leaves, while producing a zero, or minimal decrease in the ensemble response to valid classifications in the training set.

Formally, the method can be represented as the following simple minimisation problem:

$$\underset{L^*}{\text{maximize}} \quad \sum_{i=1}^{n_f} \sum_{(\mathbf{d}, y) \in \mathcal{D}} L^*_{i,d_i,y} - \lambda \|L^*\|_0 \quad (10)$$

$$\text{subject to} \quad L^* \preceq L' \quad (11)$$

The term \mathcal{D} represents the set of training samples, the entry-wise pseudo-norm $\|L^*\|_0$ evaluates the number of non-zero elements in L^* , and \preceq evaluates the *smaller-than* inequality between matrix elements. The constraint 11 ensures that the optimisation will not increase values in L . For $\lambda > 0$, with the L_0 regularisation the solution obtained will be sparse. The terms $L^*_{i,j,k}$ are maximized, so that some of the elements in L^* will still have a high value close to the original value in L' . The solution to this simple formulation can be found using a closed-form method. For a given fern number $i = n_f$ the following equivalence holds:

$$\sum_{(\mathbf{d}, y) \in \mathcal{D}} L^*_{i,d_i,y} = \sum_{j=1}^{2^{n_t}} \sum_{k=1}^{n_c} A_{i,j,k} L^*_{i,j,k}$$

Thus the objective function 10 can be rewritten as follows:

$$\sum_{i=1}^{n_f} \sum_{j=1}^{2^{n_t}} \sum_{k=1}^{n_c} A_{i,j,k} L_{i,j,k}^* - \lambda u(L_{i,j,k}^*) \quad (12)$$

where $u(\cdot)$ is the unit step function $u(x) = \{0 \forall x < 0, 1 \forall x \geq 1\}$ (or Heavyside function). This new form of the objective function shows that the optimization problem is separable. The value for each element in the optimal solution L^* can be found by solving the following reduced optimization problem:

$$\begin{aligned} & \underset{L_{i,j,k}^*}{\text{maximize}} && A_{i,j,k} L_{i,j,k}^* - \lambda u(L_{i,j,k}^*) && (13) \\ & \text{subject to} && L_{i,j,k}^* \leq L'_{i,j,k} && (14) \end{aligned}$$

For $L_{i,j,k}^* = 0$ the value for the objective function 13 becomes zero. Otherwise, it is equal to $A_{i,j,k} L_{i,j,k}^* - \lambda$. Thanks to the constraint in equation 14 and the normalisation done in equation 9, the solution for the reduced problem can only be within the range $0 \leq L_{i,j,k}^* \leq L'_{i,j,k}$. If $A_{i,j,k} L'_{i,j,k} - \lambda \leq 0$ the objective function can only produce zero, or negative values for any non-zero value of $L_{i,j,k}^*$. In these cases, the optimal solution for the reduced problem is $L_{i,j,k}^* = 0$. Otherwise, the objective function will produce the maximum value for $L_{i,j,k}^* = L'_{i,j,k}$. A summarisation of the sparsification algorithm can be found in algorithm 1.

A zero value in $A_{i,j,k}$ implies $A_{i,j,k} L_{i,j,k}^* = 0$. Hence the corresponding element $L_{i,j,k}^*$ in the leaves will have a null or negative contribution to the margin error. By setting λ to the smallest non-zero value in $A_{i,j,k} L_{i,j,k}^*$:

$$\lambda_0 = \min (\{A_{i,j,k} L'_{i,j,k}, \forall A_{i,j,k} \neq 0\}) \quad (15)$$

the optimisation will eliminate only those elements in the leaves that have zero contribution to correct classifications.

Maximizing 10 is equivalent to minimising a simplified version of the empirical loss for multiclass margin classification, such as 2 in [10]. Equation 10 represents the sum of responses to valid classes for the training samples, produced by 1, minus a L_0 regularisation. Maximising it will produce a classifier with high responses to the valid classes for the training samples. This will provide a configuration with a low value in the margin loss optimised in multiclass margin classification. By not involving the responses to incorrect classes in 10, the sparsification will not provide a classifier with optimal training error. However, it allows for a fast closed-form solution with highly accurate empirical results, as discussed in section 6.

Furthermore, for $\lambda = \lambda_0$ the training error is ensured to remain equal or smaller after the sparsification, as only those elements $L'_{i,j,k}$ for which $A_{i,j,k}$ is zero are eliminated. Each element $A_{i,j,k}$ contains the number of times the corresponding term $L'_{i,j,k}$ is added to the response of a valid

class in equation 1. If $A_{i,j,k} = 0$, the term $L'_{i,j,k}$ will never be added to the valid class response of any training sample. Decreasing them to zero will not increase the response to invalid classes for the training set, while valid responses will remain the same.

Algorithm 1 Leaves sparsification algorithm

Inputs:

$n_f, n_l, n_c \leftarrow$ Number of ferns, leaves per fern, and number of classes in the classifier.

$L', A \leftarrow$ Posterior probabilities and counts in leaves.

$\lambda \leftarrow$ Sparsification parameter.

Output:

$L^* \leftarrow$ Sparse matrix of leaves values.

```

for  $i = 1..n_f, j = 1..n_l, k = 1..n_c$  do
  if  $A_{i,j,k} L'_{i,j,k} \geq \lambda$  then
     $L_{i,j,k}^* = L'_{i,j,k}$ 
  else
     $L_{i,j,k}^* = 0$ 
  end if
end for

```

5. Efficient storage for sparse ferns

The sparse structure stores each leaf in matrix L^* as a list that contains the non-zero posterior probabilities for each class. For indexing purposes, each entry in the list must also store the class number for the value. Our implementation assumes that the ferns ensemble will emit predictions for less than 2^{16} classes. This way, the class number for each element in the list can be stored in a 16-bit integer that occupies another extra two bytes, adding up to a total of three bytes per non-zero element in L^* . In contrast, the original implementation of ferns used 4 bytes to store each value as a floating point number. Our sparse implementation must also keep an array of size $n_c \times n_{2^t}$, containing pointers to the initial position of each leaf in L^* for its fast retrieval.

When the number of zero elements is too small, sparse storage structures are known to consume more memory, and have larger retrieval times than dense ones due to the extra indexing of the sparse data. Thanks to the quantisation our sparse structure will usually require a smaller amount of memory than the original dense storage for sufficiently large classification problems, even when the sparsity in L^* is close to zero. When the sparsity is sufficiently high, the classification time is also equivalent or smaller than the time required to access the dense structure with the original dense matrix.

Sample model images.



(a) bikes (b) wall (c) graffiti

Sample images used for validation of *graffiti*.



Figure 1: Sample images for training and validation used in our tests for image patch classification.

6. Results

We performed experiments comparing classification time, prediction accuracy, and memory usage of classical ferns ensembles with and without the proposed post-processing. In this section we show the most representative results. The first subsection describes results for image patch classification, and the second results obtained for other kinds of classification problems. Results not included in these sections can be found in the additional material.

6.1. Image patch classification

We replicated the experimental setup used in [20, 19] to evaluate the performance of fern classifiers for image patch recognition. In our tests we used several model images, most of them from the Oxford¹ and CalTech² image repositories. This section shows representative results of the performance evaluation obtained for three model images: *bikes* (figure 1a), *wall* (difficult classification of features with high visual similarity, figure 1b), and *graffiti* (using real images to measure validation error, figure 1c). The results for the rest of model images are equivalent to those herein provided and can be found in the additional material.

We trained ferns ensembles using a generative procedure to recognise 400 features detected in each one of these images, using 4000 synthetically generated random affine views. This section does not include results for discriminative ferns due to the large training time in these experiments, as further discussed below. To simulate real image noise we added a Gaussian error to each pixel in these training images, and blurred them with a box filter of size 3×3 . We fixed the number of ferns in each ensemble to $n_f = 25$, and evaluated the binary tests inside an image patch of 32×32 pixels centered at each key-point location.

¹<http://www.robots.ox.ac.uk/vgg/data/data-aff.html>

²http://www.vision.caltech.edu/Image_Datasets/Caltech101/Caltech101.html

We evaluated the experiments for different numbers of tests per fern ($n_t = 10..13$). Using $n_t = 13$ we obtained results equivalent to those obtained with larger fern depths, for most data-sets in our tests. We executed the sparsification method proposed in section 4 for different increasing values of the λ parameter starting from zero. For each λ value we measured the memory size and the classifier prediction accuracy.

Prediction error vs memory consumption. Figures 2a, 2b and 2c show results of experiments where the classifiers were trained using the model images *bikes*, *wall*, and *graffiti* respectively. For the first two images the prediction accuracy was measured using 1000 random affine views of the model image, that were generated for validation purposes only. We evaluated the prediction error in the plot 2c using 40 validation images taken with a real camera. The model image was printed and captured in these validation images from different points of view (some of them can be seen at the two bottom row of figure 1).

When λ is close to zero both the memory consumption and prediction accuracy tend to be high. Conversely, for sufficiently large values of λ the memory consumption is reduced at the expense of increasing the error. The middle marker in each line shows the point where the sparsification eliminated all the elements in the leaves with zero contribution to the correct responses of the ferns ensemble for the training set. This is the point where the λ parameter is equal to λ_0 from equation 15. We obtained the peak prediction accuracy at this point for the largest number of tests per fern evaluated ($n_t = 13$ in our experiments). This peak accuracy was better than the best accuracy obtained with the original ferns ensemble in several percentage points, for all the model images used in the experiments.

As shown in figure 2a and table 3 the best prediction error obtained with sparse ferns for the model image *bikes* was 6.1%, and the ferns ensemble occupied 26.6Mb in memory. The original ferns ensemble classifier provided a classification error rate of 13.7% for a configuration that consumed a similar amount of memory (39.1Mb with $n_t = 10$). Meanwhile, the original ferns obtained a best prediction error of 9.9% for the same model image, consuming 312.5Mb of memory (with $n_t = 13$). This shows two important facts. On the one hand, redundant elements occupied a large amount of memory (89.2% for $n_t = 13$ for *bikes*, as seen in the table). Secondly, discarding these elements increases the classification success rate of the ferns ensemble (in 3.8% for *bikes*), and can provide reductions of around 90% in memory size. A similar analysis can be done for the results obtained with the rest of the model images.

We can also increase λ from this point to decrease the validation accuracy down to a certain minimally acceptable level. This way we can further reduce the memory con-

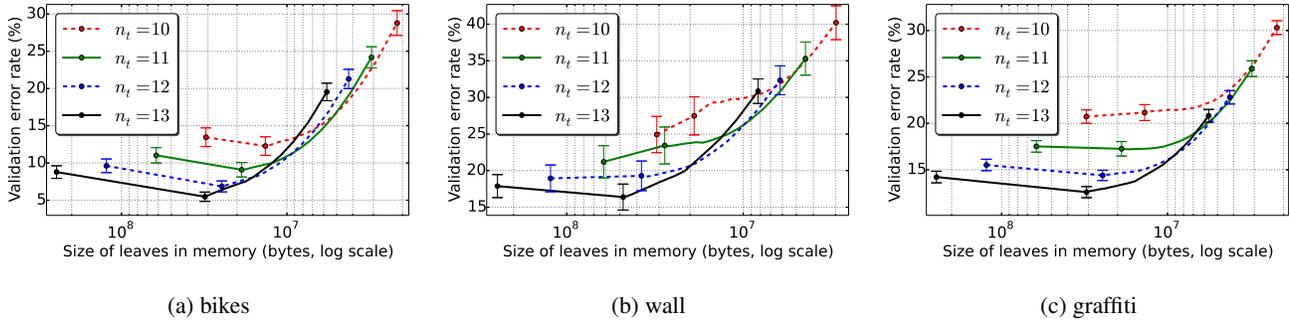


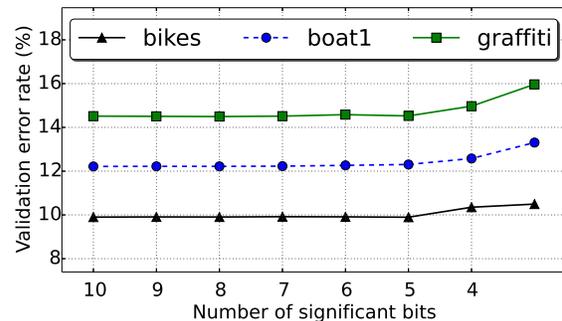
Figure 2: Comparison of memory occupied by sparse container vs. mean error and 95% confidence margin for 100 random ferns ensembles, compressed using different λ values. The second marker in each line indicates the point where $\lambda = \lambda_0$.

sumed by the classifier. As λ increases the method discards more elements in L that have positive, but small contribution to valid classification responses for the training samples. The optimisation of the error in equation 10 allowed a controlled reduction of the memory consumption, at the expense of decreasing the prediction accuracy. For example, we can configure the λ parameter to obtain a validation error with the sparse classifier similar to the error of the original ferns. This way, the memory size of the ferns ensemble trained with the *bikes* image and $n_t = 13$, is reduced down to 13.6Mb of memory, while the classifier still provides a validation error of 9.2% (smaller than the best error obtained with the original classifier). This represents an extra 50% memory reduction.

Influence of quantisation in prediction error. The plot in figure 3 shows that the quantisation produces a neglectable impact on the classifier success rate for a number of significant bits around 8. Furthermore, thanks to the quantisation and the optimal memory usage of the indexing data, storing the ferns in the sparse structure occupies a small amount of memory, even if we account for the indexing data. For these experiments our sparse container requires less memory than the dense container used in the original ferns implementation, even when L^* contains a few zero values (no discarding of elements).

In our experiments the generative training provided L matrices that contain less than 1% zero elements. Storing the quantised values in a dense container will require a 25% of the original memory size. In most of these examples, given the small number of classes for these data-sets this is less memory than using the sparse container. The quantisation did not degrade the prediction accuracy for the classification problems evaluated.

Prediction time vs memory consumption. Figure 4 compares the average time required to obtain the posterior probabilities for the image patches using the original ferns



n_t	Original ferns		Post-processed leaves		
	Error	Size	Error	Size (*)	Sparsity
10	13.7	39.1	12.3	12.0	59.5
11	11.3	78.1	8.9	16.6	72.1
12	10.2	156.3	7.0	21.6	82.1
13	9.9	312.5	6.1	26.6	89.2

Figure 3: **Top:** validation error for ferns ensembles using different quantisation levels, trained with $n_t = 13$ tests per fern using several model images. **Bottom:** error, size (in megabytes) and sparsity obtained for the model image *bikes* with the generative fern implementation, before and after the elimination of non-contributive elements with the sparsification method. (*) probabilities were quantised and stored in memory using the sparse container.

ensemble and the sparse ferns ensemble after the compression. The tests were executed on an Intel Core i3 with 3GHz and 4Gb of memory, using a single CPU core. When the sparseness is sufficiently high, the prediction time is equal or smaller than using the original classifier. In the examples previously discussed for the model image *bikes*, using the sparse container does not affect (for $\lambda = \lambda_0$) or even improves the prediction time (for larger values of λ), given the small number of non-zero elements that have to be accessed in each leaf to make the prediction.

Image	Tests	Original ferns					Post-processed leaves					
		Generative		Discriminative			Generative			Discriminative		
		Error	Size	Error	Size (*)	Sparsity	Error	Size (*)	Sparsity	Error	Size (*)	Sparsity
glass	4	36.9	9.5	36.0	9.5	31.0	34.6	5.0	64.0	35.0	5.0	64.0
	6	32.7	37.6	33.6	37.6	58.4	30.8	13.7	85.0	31.8	13.7	85.0
	8	33.6	150.1	32.7	150.1	83.8	30.8	43.0	95.2	33.2	43.0	95.2
ionosphere	4	16.5	3.2	17.1	4.8	7.6	15.1	4.6	8.5	16.2	4.6	8.5
	6	12.5	12.6	12.3	18.9	29.1	10	15.1	40.4	10.0	15.1	40.4
	8	13.7	50.1	12.5	75.1	63.3	12.3	46.9	75.3	12	46.9	75.3
wine	4	7.3	4.8	9.6	6.0	14.1	7.3	4.5	43.2	6.7	4.5	43.2
	6	4.5	18.8	6.2	23.6	46.0	3.9	13.2	73.8	4.5	13.2	73.8
	8	3.4	75.1	3.9	93.9	77.0	3.9	42.9	90.5	2.8	42.9	90.5
ecoli	4	21.4	12.6	30.7	11.8	41.6	20.5	4.8	74.9	22.0	4.8	74.9
	6	18.2	50.1	18.2	47.0	68.4	16.7	13.0	90.7	16.1	13.0	90.7
	8	15.5	200.1	16.1	187.6	85.1	15.8	42.7	96.7	14.6	42.7	96.7

Table 1: Comparison of memory size (in kilobytes), leave-one-out cross-validation error, and sparsity for different classification problems. The table shows the results obtained using generative and discriminative ferns ensembles, before and after the post-processing. (*) probabilities were quantised and stored in memory using the sparse container.

In our experiments the generation of the matrix A in the training usually required between 3 and 4 seconds. Depending on the number of tests per fern n_t , obtaining the matrix L of posterior probabilities with the generative ferns training took between 1 and 12 seconds, and the sparsification took between 0.5 and 2 seconds.

Each regularised Hinge loss optimisation in the discriminative training involved more than a million samples and 2^{10} variables, taking more than a minute to find the solution using a standard sparse SVM implementation. Training a single discriminative ferns ensemble would hence require more than $400 \times 25/60 \simeq 160$ hours for the current setup used in our tests for image patch classification. For this reason this section does not include results for discriminative ferns. Next section shows results with smaller classification problems comparing generative and discriminative ferns.

6.2. General classification problems

Table 1 shows results of experiments where we used several publicly available data-sets from the UCI repository³. Each one of these data-sets contains the samples and labels for a different kind of classification problem. These data-sets have less than 1000 samples and 10 classes each. The features in these data-sets were normalised for the experiments, so they have zero mean and unit variance.

The size for the discriminative ferns shown in table corresponds to quantised values that were stored in the sparse container. This is done to compare the memory occupied by the discriminative classifier before and after our sparsification, when it is stored in the sparse container. The discriminative training produced a significant number of zero

elements. However, our method provided a much higher sparsity.

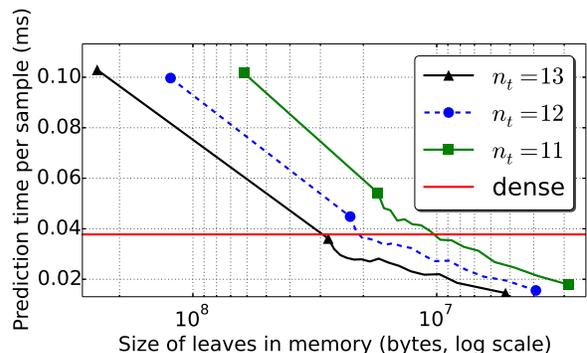


Figure 4: Average time per prediction with generative ferns before and after the post-processing. The classifier was trained using the model image *bikes*.

7. Conclusions

We demonstrated that ferns ensembles typically contain a large number of elements that can be discarded without harming prediction accuracy. Likewise we can quantise the remaining elements using a reasonable bit depth and have a high classification performance. We provided empirical results showing the large memory reductions achieved by exploiting these facts for generative and discriminative ferns ensembles. In the future, optimisations like the present will permit new applications using low-end hardware with strict memory restrictions, such as visual navigation and recognition using micro-robots or light-weight wearable devices.

³<http://archive.ics.uci.edu/ml/>

References

- [1] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural computation*, 9(7):1545–1588, 1997. **1**
- [2] D. Anguita and G. Bozza. The effect of quantization on support vector machines with Gaussian kernel. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, volume 2, pages 681–684 vol. 2, 2005. **2**
- [3] J. Bi, K. Bennett, M. Embrechts, C. Breneman, and M. Song. Dimensionality reduction via sparse support vector machines. *Journal of Machine Learning Research*, 3:1229–1243, 2003. **2**
- [4] A. Bosch, A. Zisserman, and X. Munoz. Image classification using random forests and ferns. In *ICCV*, 2007. **1**
- [5] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. **1**
- [6] M. Calonder, V. Lepetit, P. Fua, K. Konolige, J. Bowman, and P. Mihelich. Compact signatures for high-speed interest point description and matching. In *ICCV*, 2009. **1, 3**
- [7] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. Brief: Binary robust independent elementary features. In *ECCV*, pages 778–792, 2010. **3**
- [8] X. Cao, Y. Wei, F. Wen, and J. Sun. Face alignment by explicit shape regression. *International Journal of Computer Vision*, 2013. **1**
- [9] C. Caraffi, T. Vojir, J. Trefny, J. Sochman, and J. Matas. A System for Real-time Detection and Tracking of Vehicles from a Single Car-mounted Camera. In *IEEE 15th International Conference on Intelligent Transportation Systems (ITSC)*, pages 975–982, 2012. **1**
- [10] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2002. **1, 2, 5**
- [11] B. Glocker, J. Shotton, A. Criminisi, and S. Izadi. Real-time RGB-D camera relocalization via randomized ferns for keyframe encoding. *IEEE Trans. on Visualization and Computer Graphics (TVCG)*, 2014. **1, 3**
- [12] K. Konolige. Sparse sparse bundle adjustment. In *BMVC*, pages 102.1–102.11, 2010. **2**
- [13] E. Krupka, A. Vinnikov, B. Klein, A. Bar Hillel, D. Freedman, and S. Stachniak. Discriminative ferns ensemble for hand pose recognition. In *CVPR*, 2014. **1, 2, 4**
- [14] V. Lepetit. Keypoint recognition using randomized trees. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 2006. **1**
- [15] V. Lepetit and P. Fua. Keypoint Recognition using Random Forests and Random Ferns. In A. Criminisi and J. Shotton, editors, *Decision Forests for Computer Vision and Medical Image Analysis*, number 9, pages 111–124. 2013. **1**
- [16] B. Lesser, M. Mcke, and W. N. Gansterer. Effects of reduced precision on floating-point SVM classification accuracy. *Proceedings of the International Conference on Computational Science (ICCS)*, 4(0):508 – 517, 2011. **2**
- [17] T. Luo, L. O. Hall, D. B. Goldgof, and A. Remsen. Bit reduction support vector machine. In *Proc. of IEEE International Conference on Data Mining*, pages 733–736, 2005. **2**
- [18] J. Marin, D. Vazquez, A. M. Lopez, J. Amores, and B. Leibe. Random forests of local experts for pedestrian detection. In *ICCV*, 2013. **1, 2**
- [19] M. Özuysal, M. Calonder, V. Lepetit, and P. Fua. Fast keypoint recognition using random ferns. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 32(3):448–461, 2010. **1, 2, 6**
- [20] M. Özuysal, P. Fua, and V. Lepetit. Fast keypoint recognition in ten lines of code. In *CVPR*, 2007. **1, 3, 4, 6**
- [21] A. G. Schwing, C. Zach, Y. Zheng, and M. Pollefeys. Adaptive random forest - how many "experts" to ask before making a decision? In *CVPR*, pages 1377–1384, 2011. **2**
- [22] P. Sharma and R. Nevatia. Efficient detector adaptation for object detection in a video. In *Proc. Computer Vision and Pattern Recognition*, pages 3254–3261, 2013. **1**
- [23] Y. She. Thresholding-based iterative selection procedures for model selection and shrinkage. *Electron. J. Statist.*, 3:384–415, 2009. **2**
- [24] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *CVPR*, pages 1297–1304, 2011. **1**
- [25] T. Trzcinski, M. Christoudias, P. Fua, and V. Lepetit. Boosting binary keypoint descriptors. In *CVPR*, pages 2874–2881, 2013. **3**
- [26] T. Tuytelaars and K. Mikolajczyk. Local invariant feature detectors: A survey. *Foundations and Trends in computer graphics and vision*, 3(3):177–280, 2008. **3**
- [27] M. Villamizar, F. Moreno-Noguer, J. Andrade-Cetto, and A. Sanfeliu. Shared random ferns for efficient detection of multiple categories. In *ICPR*, 2010. **1**