Discriminative Ferns Ensemble for Hand Pose Recognition

Eyal Krupka, Alon Vinnikov, Ben Klein, Aharon Bar Hillel, Daniel Freedman
Microsoft Research
Haifa, Israel
eyalk,alvinn,t-benk,aharonb,danifree@microsoft.com

Simon Stachniak
Microsoft Console dev R&D
Redmond, USA
szimons@microsoft.com

Abstract

We present the Discriminative Ferns Ensemble (DFE) classifier for efficient visual object recognition. The classifier architecture is designed to optimize both classification speed and accuracy when a large training set is available. Speed is obtained using simple binary features and direct indexing into a set of tables, and accuracy by using a large capacity model and careful discriminative optimization. The proposed framework is applied to the problem of hand pose recognition in depth and infra-red images, using a very large training set. Both the accuracy and the classification time obtained are considerably superior to relevant competing methods, allowing one to reach accuracy targets with run times orders of magnitude faster than the competition. We show empirically that using DFE, we can significantly reduce classification time by increasing training sample size for a fixed target accuracy. Finally a DFE result is shown for the MNIST dataset, showing the method's merit extends beyond depth images.

1. Introduction

The tradeoff of speed versus accuracy is an important topic, widely discussed in the object detection and recognition literature [11, 8, 3, 18, 22]. In applications like Natural User Interface (NUI), algorithms have to obtain high recognition accuracy in real time, on low power platforms. Often accuracy must be obtained with only a small fraction of the available CPU resources, reserving CPU cycles for other operations. The tradeoff is natural: high accuracy requires a rich representation, with considerable computational cost at all levels of the system. At the lowest level, this includes using dense sampling of complex local descriptors [19, 23]. Further on, multiple spatial aggregation layers are employed [16, 6], with large dictionaries at higher levels. At the highest level the best accuracy is often obtained using non-linear kernels [16, 6], requiring kernel computation with many support vectors.

In this work we attack the problem of hand pose classification using infra-red (IR) and depth images from a time of flight depth camera, in the context of a NUI application. There are dual demands for high accuracy and a very low computation budget - a fraction of a millisecond. For our problem, standard techniques achieved reasonable enough accuracy for a moderate training set size, but were unable to meet the classification time requirement. One can improve speed by modifying the parameters of such techniques, like grid density or dictionary size. However, experiments show that this approach is limited: when the target speed is obtained accuracy drops too much.

This calls for a wider consideration of recognition systems based on machine learning. Beyond accuracy and speed, these systems have additional performance characteristics: generalization ability (i.e. ability to learn from a relatively small training set size), training time, and memory consumption. Suppose that it is possible to collect a very large training set, there is no significant limitation on training time, and a moderate amount of memory is available at test-time. The question then becomes: for a fixed accuracy target, can we trade training set size for increased speed at test time?

We propose pushing the speed-accuracy envelope at the expense of larger training sets using three steps. First, we use simple non-invariant features with sharp non-linearity, as they are fast to compute. Using a large enough training set, we can hope that the task-relevant invariance will be learned instead of *a priori* encoded. Second, and most important, we turn to an architecture with large capacity and minimal computation, based on an ensemble of large

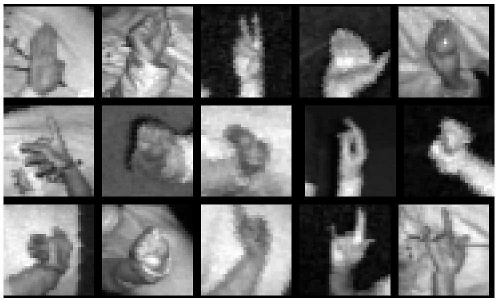

Figure 1. Examples of hand images from our data set. The left columns contain examples of 'Open' ,'Closed' and 'Lasso' respectively. The two right columns contain examples of the complement class 'Other'.

tables encoding the end results. Such table-based classifiers, termed 'ferns' [6, 21, 18], have high capacity with a VC-dimension of 2^K for a single 2^K-entry table, and close to $M2^K$ for a M-tables ensemble[1]. Third, since we face a hard learning problem, with high capacity and minimal prior, we develop a discriminative optimization framework for a fern ensemble, which is a departure from the generative formulation used previously for ferns.

Focusing on speed optimization, we use as features spatial aggregates of highly simplistic features, i.e. pixel-pair comparisons; we then build a lookup table (fern) from a set of such bit features. Instead of a huge single table, we learn a fern ensemble. Each fern is based on a set of K simple binary features and a large table of 2^K-entries. The binary features are concatenated into an index, and the corresponding index entry in the table contains a weight contribution, summed across the ferns to get the final classification. Each table can be regarded as an efficient codeword dictionary: it maps a patch into one of 2^K words, yet at the cost of K operations. The resulting architecture is highly non-linear, and a feed-forward push of an image through it only requires multiple bit computations and table access operations.

Ferns are traditionally formulated generatively, i.e., conditional class probabilities are stored at the table entries. In contrast, we train the ensemble discriminatively by minimizing the regularized hinge loss, i.e., the loss minimized by Support Vector Machines (SVM). The minimization technique is related to ideas from the Predictive Feature Selection (PFS) algorithm [2]. It is done agglomeratively in a boosting-like framework, promoting complementariness between chosen ferns and between bits in a single fern.

Our main technical contribution is hence in the introduction of a *Discriminative Ferns Ensemble (DFE)* approach, and empirically demonstrating its ability to considerably shift the speed-accuracy curve. The method is applied to hand pose recognition from IR and depth images, and achieves accuracy comparable or better than the best known methods while being one to two orders of magnitude faster. Specifically it is significantly more accurate than a classification based on deep random trees, which have been used for similar tasks [15, 22], and considerably more accurate than a more standard ensemble of random ferns [6, 21]. We also tested several methods combining fast dense SIFT features, DAISY, random forest dictionaries, and SVM [20, 26, 25]. The best results achieved were slightly less accurate than DFE, but classification time was two orders of magnitude (i.e. 100 times) slower.

The second contribution is that we empirically show significant improvements in classification speed – for a given target accuracy – can be achieved by collecting larger training sets. This is done by optimizing K (log of the table size) and M (number of ferns) for a given training set size. In other words, if a DFE classifier is accurate, but not fast enough, collecting larger training set can be used to accelerate classification speed. Note this trade-off is different from the known trade-off between training set size and accuracy.

Finally, we note that our approach was found practical and recently integrated into a real commercial system.

We discuss related work in Section 2, and present our approach in Section 3. In Section 4 we summarize a set of experiments in which ingredients of the method are tested and the approach is compared with competing techniques. We briefly conclude in Section 5.

2. Related Work

The methods most similar to our approach are those based on trees and ferns ensembles [22, 15, 21, 6, 12, 9]. Ferns are often regarded as a special case of trees, in which the condition encoded at all the nodes with the same depth is identical. Boosting of decision trees is a highly popular technique [12, 9], but usually shallow trees of depth $1-3$ are used, which cannot capture fine-grained partitions. Ferns ensembles were suggested in recent years for image classification [6], keypoint recognition [21] and nearest neighbor finding [18]. In these works ferns in the ensemble are chosen independently of each other, and bits in a single fern are chosen at random or using an information gain criterion. At the leaves conditional class posteriors are computed and averaged across ferns [6], or regulated with a prior and multiplied [21]. In our approach we regard the fern ensemble as providing the features for a large L_2-SVM problem. Ferns are added to the ensemble one-by-one in an agglomerative procedure. The gradient of the SVM program w.r.t adding new features is computed at each round and used to guide choice of the bits in the new fern. Ferns (and bits) are hence grown to be complementary, maximally reducing a linear approximation of the loss at time of their addition. The weights at the fern's table (corresponding to the leaves of a tree) are optimized using an SVM program applied to a set of $M2^k$ sparse features.

Classification and pose estimation in IR+depth images

[1] Assuming that the underlying space is of dimension higher than K and MK respectively, which are satisfied for the image sizes we consider

Algorithm 1 Ferns Ensemble: Classification

```
Input: An image I of size $S_x \times S_y$,
classifier parameters $(B^m, A^m, W^m)_{m=1}^M$, threshold $t$
$B^m \in \mathbb{R}^{K\times|A^m|}$, $A^m \subset \{1,..,S_x\} \times \{1,..,S_y\}$, $W^m \in \mathbb{R}^{2^K}$
Output: A classifier decision in $\{0,1\}$
Initialization: Score=0
For all ferns $m = 1,..,M$
    For all pixels $p \in A^m$
        Compute a k-bit index = $\sigma(B^m I_{N(p)})$
        Score=Score+$W^m$[index]
Return (Score>t)
```

was addressed in recent years in multiple works [22, 15, 5, 1, 10]. For a survey of earlier work on hand recognition see [13]. Among these, the works presented at [22, 15] are similar to ours, as they also turn to simple pixel-comparison features and a tree ensemble architecture, to allow fast, real time classification. However, they use a generatively-optimized random forest approach, which is less accurate, and requires more memory and CPU time than the proposed method. We discuss the differences in section 3.2, and compare the methods empirically in section 4. From a different perspective, our work shares with some earlier work like [7] the motivation to balance speed and accuracy. However cascade-based solutions like discussed in this work are different, and actually complementary to our approach.

3. The Discriminative Ferns Ensemble

We describe the Fern Ensemble classifier in Section 3.1 and analyze its running time in Section 3.2. In Section 3.3 we present the training procedure we use.

3.1. The Discriminative Ferns Ensemble Classifier

The ferns ensemble classifier operates on an image patch, which we denote by I, consisting of P pixels. For a pixel p, we denote its neighborhood by $N(p)$, and we denote by $I_{N(p)}$ the subpatch which is comprised of the pixels in p's neighborhood. In what follows, we will consider $I_{N(p)}$ as a vector in $\mathbb{R}^{|N(p)|}$. The ferns ensemble consists of M individual ferns, and its pipeline includes three layers whose structure we now describe.

Bit Vector Computation Let us focus on one particular fern m. For each pixel p, we compute a local descriptor of its neighborhood subpatch $I_{N(p)}$ using computationally-light pairwise pixel comparisons of the form

$$I_{q_1} \overset{?}{>} I_{q_2} \quad \text{for } q_1, q_2 \in N(p) \tag{1}$$

Such a comparison provides a single bit value of 0 or 1. For convenience of notation, we may rewrite the bit obtained as $\sigma(\beta^T I_{N(p)})$, where β is a $|N(p)|$-dimensional sparse vector, with two non-zero values, one equalling 1, the other equalling -1; and σ is the Heaviside function. For each fern m and pixel p, there are K bits computed, and we denote the k^{th} bit as $b_{p,k}^m = \sigma((\beta_k^m)^T I_{N(p)})$. Collecting all the bits together, the K-dimensional bit vector b_p^m is:

$$b_p^m = \sigma(B^m I_{N(p)}) \in \{0,1\}^K \tag{2}$$

where the matrix B^m has rows $(\beta_1^m)^T, \ldots, (\beta_K^m)^T$; and now the Heaviside function σ is applied element-wise.

Histogram of Bit Vectors We are interested in some translation invariance, so we take a spatial histogram over codewords. However, as in [21] the bit-vectors *themselves* are the codewords; there is no need for an intermediate clustering step. Denote the histogram for the m^{th} fern by $H^m(b)$, where bit vector $b \in \{0,1\}^K$; then

$$H^m(b) = \sum_{p\in A^m} \delta(b_p^m - b) \tag{3}$$

where δ is a discrete delta function, and $A^m \subset \{1,..,P\}$ is the spatial aggregation region for fern m. Note that H^m is a sparse vector, with at most P non-zero entries.

Histograms concatenation The final decision is made by a linear classifier applied to the concatenation of the M fern histograms.

$$f(I) = W^T H(I) = \sum_{m=1}^{M} \sum_{b\in\{0,1\}^K} w_b^m H^m(b) \tag{4}$$

where $H(I) = [H^1(I), \ldots, H^M(I)] \in \mathbb{N}^{M2^K}$ and $W = [W^1, \ldots, W^M] \in \mathbb{R}^{M2^K}$ is a weight vector. Combining Steps 1-3 in the pipeline, we arrive at the Discriminative Ferns Ensemble Classifier:

$$f(I;\rho) = \sum_{m=1}^{M} \sum_{b\in\{0,1\}^K} w_b^m \sum_{p\in A^m} \delta(\sigma(B^m I_{N(p)}) - b) \tag{5}$$

with the parameters $\rho = \{W^m, B^m, A^m\}_{m=1}^M$.

3.2. Classification Speed

Algorithm 1 describes the operation of a DFE classifier at test time. The pipeline is extremely simple. For each fern and each pixel in the fern's aggregation region we compute the bit vector, considered as a codeword index. The fern table is then accessed with the computed index, and the obtained weight is added to the classification score. The complexity is O$\big(M\overline{A}K\big)$ where $\overline{A}$ is the average number of pixels per aggregation region: $\overline{A} = \frac{1}{M}\sum_m |A^m|$.

It is interesting to compare the CPU time of a single fern to a single tree with the depth K. From a pure computational complexity perspective, the number of operations for both is K. Nevertheless, a closer look at their match to common CPU architectures, including cache hierarchies and vector machines, reveals large differences in expected

Algorithm 2 Ferns Ensemble: Training

```
Input: A labeled Training set $\{I_i, y_i\}_{i=1}^N$
       Parameters $M, K, C, N_c, \{A^m\}_{m=1}^M$
Output: A classifier $(B^m, A^m, W^m)_{m=1}^M$, threshold $t$
Initialization: $Z[i] = 1/|\{I_i|y_i = 1\}|$  if $y_i = 1$,
                $Z[i] = -1/|\{I_i|y_i = -1\}|$ if $y_i = -1$
For $m = 1, .., M$
    For $k = 1, .., K$
        For $c = 1, ..N_c$
            Sample a candidate column $\beta^m_{k,c} \in R^{|N(p)|}$
            For $i = 1, .., N$
                Compute $H^m(b, I_i, c) = H^m(b, I_i; B^m_c)$
                 with $B^m_c = [\beta^m_1, .., \beta^m_{k-1}, \beta^m_{k,c}]$
            For $b \in \{0,1\}^K$
                Compute $R_Z(c) = \sum_{b\in\{0,1\}^K} R_Z(H^m(b; \beta^m_{k,c}))$
        Choose winning candidate $c^* = argmax_c R(c)$,
        and set $\beta^m_k = \beta^m_{k,c^*}$
    Train an SVM with $m2^K$ features $W \cdot [H^1, .., H^m] - t$
    Set $Z[i] = y_i\alpha_i$ for $i = 1, .., N$ with $\alpha_i$ SVM dual variables
Set $\{W^m\}_{m=1}^M, t$ based on the last SVM training.
Return $(B^m, A^m, W^m)_{m=1}^M$, threshold $t$.
```

run time. First, a tree needs to store the bit computation parameters for 2^K internal nodes. More importantly, during tree traversal the working set is accessed K times in an unpredictable manner. A fern's operation requires only a single access to its large working set (W^m), as the index computation is done using a small amount of memory, $O(K)$ in size, which fits in the cache without a problem.

Second, the usage of fixed pixel pairs in a fern enables computation of the K-bit index without indirection and with an unrolled loop. More importantly, ferns are amenable to vectorization using SIMD (Single Instruction, Multiple Data) operations, while trees are not. Applying a fern operation to several examples at the same time (i.e. vectorizing the loop over p in Algorithm 1) is straightforward. Doing so for a tree is likely to be extremely inefficient since each example require a different sequence of memory accesses, and gathering such scattered data cannot be done in parallel in a SIMD framework. In Section 4.4 we further discuss the differences of ferns and random forest, in terms of classification time and memory.

3.3. Discriminative Training

The DFE classifier $f(I; \rho)$ is given in Equation (5), and we would like to learn the parameters $\rho = \{W^m, B^m, A^m\}_{m=1}^M$ from a labeled training set $\{(I^i, y^i)\}_{i=1}^N$. Unlike prior work on ferns, e.g. [21], we turn to a discriminative rather than a generative formulation. Specifically, we pose the problem as regularized Hinge-loss minimization, similar to standard SVM:

$$\min_\rho \; \frac{1}{2}\|W\|^2 + C\sum_{i=1}^N \left[1 - y^i f(I^i; \rho)\right]_+ \tag{6}$$

where $[\cdot]_+$ indicates the hinge loss, i.e. $[z]_+ = \max\{z, 0\}$. Rewriting Equation (4) with explicit parameter and image dependence one gets

$$f(I; \rho) = \sum_{m,b} w^m_b H^m(b, I \;\; ; B^m, A^m) \tag{7}$$

We can see that f is linear in W, so optimizing (6) w.r.t W for fixed $\{B^m, A^m\}_{m=1}^M$ is a standard SVM optimization. However, optimizing for the latter parameters is challenging, specifically since they are to be chosen from a large discrete set of possibilities. Hence, we turn to an agglomerative approach in which we greedily add ferns one at the time. As can be seen from Equation (5), adding a single fern amounts to an addition of 2^K new features to the classifier. In order to do that in an sensible manner, we extend known results for the case of a single feature addition [2, 4].

Let $f(I) = \sum_{l=1}^{L-1} w_l x_l(I)$ be a linear classifier optimized with SVM and $L(f, \{I_i, y_i\}_{i=1}^N)$ the hinge loss obtained for it (Eq. (6)) over a training set. Assume we add a single feature x^L to this classifier $f^{new}(I) = f^{old}(I) + w_L x^L(I)$, with small $|w_L| \leq \epsilon$. Theorem 1 in [2] gives a linear approximation of the loss under these conditions:

$$L(f^{new}) = L(f^{old}) - w_L \sum_{i=1}^N \alpha_i y_i x_i^L + O(w_L^2) \tag{8}$$

where α_i are the example weights obtained as a solution to the dual SVM problem. The weights $\alpha_i \in [0, C]$ are only non-zero for support vectors. For a candidate feature x_L, the approximated loss (8) is best reduced by choosing $w_L = \epsilon \cdot sign(\sum_{i=1}^N \alpha_i y_i x_i^L)$, and the reduction obtained is $R(x_L) \triangleq |\sum_{i=1}^N \alpha_i y_i x_i^L|$. The PFS algorithm [2] is based on training SVM using a small number of features, followed by computing the score $R(x)$ for a large number of unseen features; this allows one to add/replace existing features with promising feature candidates. Note that the score $R(x)$ of a feature column x can be seen as the correlation $R_Z(x) = x \cdot Z$, where $Z = (z_1, .., z_n)$ with $z_i = y_i \alpha_i$ is the vector of signed example weights.

Here we extend the aforementioned idea to a set of features, as introduced by a single fern. Assume we have trained an SVM classifier over a fern ensemble $f^{M-1}(I)$ with $M - 1$ ferns, and we now wish to extend to an additional fern. Assume further that the new weight vector is small with $||w^m||_\infty \leq \epsilon$. Then we have

$$f^M(I) = f^{M-1}(I) + \epsilon \sum_{b\in\{0,1\}^K} w^m_b H^m(b, I) \tag{9}$$

with $|w^m_b| \leq 1$ for all b. Treating the new fern contribution as a single feature, we can apply the theorem stated above and get

$$L(f^M(I)) \approx L(f^{M-1}) - \epsilon \sum_{i=1}^N \alpha_i y_i \sum_{b\in\{0,1\}^K} w^m_b H^m(b, I_i)$$

$$= L(f^{M-1}) - \epsilon \sum_{b \in \{0,1\}^K} w_b^m \sum_{i=1}^{N} \alpha_i y_i H^m(b, I_i) \quad (10)$$

where the approximation in the first equation is due to omission of $O(\epsilon^2)$ terms. If we wish to minimize the approximated loss, the optimal choice for w_b^m is $w_b^m = sign(\sum_{i=1}^N \alpha_i y_i H_t^m(b, I_i))$, in an analogous way to the single feature case. With these w_b^m we get

$$L(f^M(I)) \approx L(f^{M-1}) - \epsilon \sum_{b \in \{0,1\}^K} R(H^m(b)) \quad (11)$$

This result is an intuitive extension of Theorem 1 in [2] for the case of multiple feature addition.

Our algorithm for fern ensemble growing is based on iterating between SVM training and building the next fern based on Equation (11). This procedure is described more precisely in Algorithm 2. At each fern addition step we use an SVM classifier trained on the previous ferns to get signed example weights, in a manner similar to boosting. The ensemble score $\sum_{b \in \{0,1\}^K} R_Z(H^m(b))$ is used to grow the fern bit-by-bit in a greedy fashion. At each bit addition stage we randomly select N_c candidates for the mask β_k^m, termed $\beta_{k,c}^m$; each candidate is chosen by randomly drawing the two pixels needed for the comparison. The winning bit is chosen as the one producing the highest ensemble score. We currently do not optimize the integration area variables $\{A_m\}_{m=1}^M$, but we experiment with several choices in Section 4. The algorithm is presented for a single binary problem, but is easily extended to training of several classes with shared A^m, B^m and separate W^m. During optimization, multiple SVMs are trained at each fern addition, and $R(c)$ scores of all of them are summed to make the bit choice.

4. Empirical Results

The method described in this paper was developed, tested and compared to alternatives on a very large data set for hand shape recognition. We describe the data set in Section 4.1 and the method's implementation details in 4.2. The impact of the main ingredients and parameters of the method is tested in 4.3. We compare the accuracy-speed trade-off enabled by the proposed method and various competing techniques in 4.4. Trade-offs between accuracy, classification time, training sample size and memory are discussed in 4.5. Finally we show a good result of the DFE on the MNIST dataset [17].

4.1. Data Set

The task we consider is to recognize three different hand shapes, and to discriminate between them and other undefined hand states. The recognition results are used as part of a NUI interface. The shapes are termed 'Open', 'Closed', 'Lasso' and 'Other', as shown in Figure 1. The class 'Other' includes a large variation in hand poses, including hands holding objects. Hand detection is achieved by tracking the skeleton in a sequence of depth+IR images, using methods based on [22].

The images used for recognition are cropped around the extracted hand position, rotated and scaled to two 36×36 images of the depth and IR channels. A simple pre-processing rejects IR and depth pixels where the depth is clearly far beyond the hand, thereby removing some of the background. The alignment and rotation of the hand is based on estimated wrist position and is sometimes inaccurate, making the recognition task harder.

A dataset of 519,000 images was collected and labeled from video sequences of different people, with $80,000$ examples made publicly available[2]. Images have considerable variability in terms of viewpoints, hand poses, distances and imaging conditions. The images were taken at distances of up to 4 meters from the camera, where the quality of image drops, and the depth measurement of fingers may be missing. Data was divided into training and test sets with $420,000$ and $99,000$ images respectively, such that persons from the training set do not appear in test images and vice versa. The data was collected to give over-representation to hard cases. Given the data properties, the goal was to achieve 2-5% false negative rate, at a false positive rate of 2%. Since the the test data is hard, the error rate in real usage scenarios is expected to be much lower.

4.2. Implementation Details

In our experiments we tested the number of bits per fern K in the range of $[3, 18]$, and the number of ferns M in $[6, 768]$. At each bit addition step $N_c = 40$ pixel comparison features were randomly generated for evaluation. The spatial aggregation area of the fern A_m was randomly chosen to be one of the 4 standard quadrants of the image patch, and the neighborhood $N(p)$ is 17×17 pixels. We have experimented with limiting the aggregation area A_m further by imposing a virtual checkerboard on the quadrant pixels: for odd bit indices features are only computed for 'white' pixels, and for even indices features are computed only for 'black' ones. This policy was found to be useful in terms of accuracy-speed trade-offs.

We have used the LibLinear package [14] for sparse SVM training of our models. The classifier was implemented in C and running times are reported on Intel core i7, 2.6GHz CPU, using a single thread. Computation time is reported for a single image in milliseconds, without usage of SIMD optimizations. Accuracy of a single binary classifier, i.e. one hand pose versus all, is computed as the false negative error rate at the working point providing a false

[2] http://research.microsoft.com/en-us/projects/ThreeHandPose/default.aspx

Pipe variation	% FN @ FP=2%
Baseline DFE	2.18
Single aggregation area	3.15
No checkerboard sampling	2.42
Naive Bayes + Boosting	3.87
Plain linear SVM	34.1
Naive Bayes, MI bits	35.9
Naive Bayes, Rand bits	47.6
Only Depth	4.65
Only IR	5.23

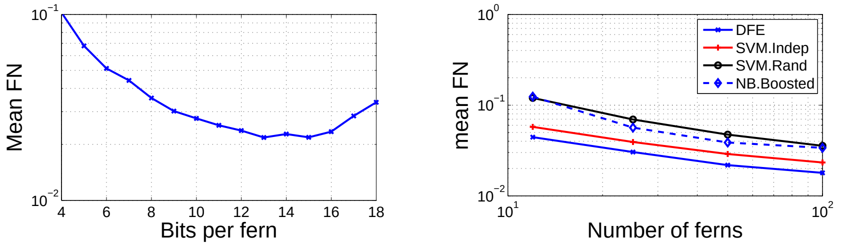

Figure 2. **Variations and parameters. Left:** Error for several DFE and Ferns algorithm variations. See text for explanation. The result of plain linear SVM applied to depth+IR pixels is also given as a baseline. **Middle:** False negative rate of the DFE as a function of K. **Right:** False negative rate as a function of M for several training procedures. DFE is our baseline variation. For both SVM.Indep and SVM.Rand, SVM is used as final classifier. For SVM.Indep the bits are selected using $R(c)$ score, but without PFS weight update, i.e. using initial $Z[i]$ for all ferns (see Algorithm 2). For SVM.Rand bits are randomly selected. NB.Boosted is Naive-Bayes with fern boosting and entropy-gain bit choice.

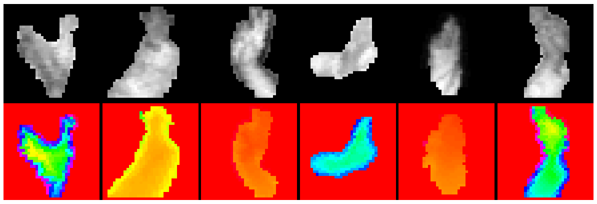

Figure 3. **Successes and failures of the DFE classifier:** Pairs of depth+IR images are presented, where the top row shows the IR images and the bottom the depth images in every pair. The 3 pairs on the left show successfully classified pairs for the 3 hand shape classes considered (Open, Closed, Lasso). The pairs on the right show miss-classification errors (false negatives).

positive (FP) rate of 2%. Accuracy figures reported here are averaged over the three classes. We selected this approach rather than multi-class error rate, as in each specific NUI usage context the three classification scores are combined in a different way.

4.3. Parameters and Variations

Success and failure examples of the DFE classifier can be seen at Figure 3. We now concentrate on understanding the contribution to performance of algorithm components.

Complexity of layers 1: At the first layer we encode patches into codeword indices, and its complexity is controlled by the number of bits K used for the encoding. In Figure 2 (middle) the classifier accuracy is plotted as a function of K for fixed $M = 50$. Based on this graph, we select the value of $K = 13$ in our subsequent experiments, as it is the minimal value which yet provide close to optimal accuracy.

Complexity of layers 2: At the second, spatial aggregation layer, complexity is controlled by several algorithmic choices. First, we can use multiple aggregation areas, or a single aggregation area containing the whole image for all ferns. Second, we can use or avoid using the checkerboard technique for computational saving. Results are reported in Figure 2 (left). Baseline DFE uses $M = 50$ ferns with quadrant ferns, checkerboard policy. The number of ferns used in the conditions 'single area' and 'no checkerboard' are reduced by a factor 4 and 2 to get classifiers with approximately the same speed as the the baseline. The results show the advantage of baseline DFE over alternatives, hence led to its definition as 'baseline'.

Complexity of layer 3, optimization policy: Figure 2 (left) shows the accuracy for several ensemble training strategies. The simpler alternatives uses Naive Bayes, where the leaf weights are based on class posterior probabilities [6, 21]. The ferns are trained independently, with bits chosen at random (Naive Bayes, Rand bits) or by maximization of information gain (Naive Bayes, MI). For these alternatives, the false negative rate is high[3]. Also, further increasing of the number ferns does not help as much as in the DFE or boosting framework, as the ferns are learned independently. Another alternative is training complementary ferns by boosting, with bits chosen to maximize the information gain on the boosting-reweighted sample (Naive Bayes + Boosting). This significantly improves accuracy relative to MI and random selection, but is still less accurate than DFE. Figure 2 (right) shows the effect of number of ferns, M, on the false negative rate for selected methods.

From the above results, we can conclude that using discriminative (SVM) approach for both the final classifier and selecting of the fern bits, significantly improves accuracy.

The table in Figure 2 also shows that IR and depth are not redundant, and using both of them significantly improves accuracy relative to using only one of them.

4.4. Speed-Accuracy Trade-Off Comparison

We have compared the fern ensemble method to several alternative architectures, which also have an emphasis on a good speed-accuracy trade-off. The methods compared are:

- Random forest applied to pixel comparisons as suggested by [15]
- A 3-stages pipeline: a) Fast dense SIFT features computation using the VLFeat library [24]. b) Encoding

[3] Note that FN is measured at false positive rate of 2%. Hence, FN near 50% is far better than random. At FP=10% the false negative rates of Naive Bayes MI bits and Rand bits drops to 11% and 18% respectively.

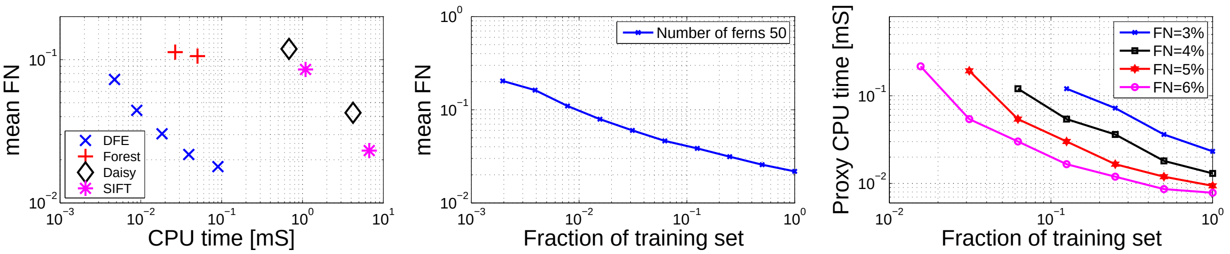

Figure 4. **Speed-accuracy trade-offs and comparison to other methods. Left:** Best results of false negative rate under constraint of classification CPU time for various methods and parameters for each method. For DFE, we modified values of M, K. For random forest [15], the points shown are for one and two trees of depth 21. Fast SIFT can achieves accuracy comparable to DFE, but at cost of more than $\times 100$ classification time. **Middle:** Accuracy obtained by DFE as a function of training sample size. X axis is the fraction of training set size relative to the full set (420,000 images). **Right:** The classification CPU time, as a function of training sample size. This is measured for several target false negative rates (for a fixed FP=2%).

into a bag of features using a random forest dictionary [20]. c) SVM classification with a linear approximation of the histogram intersection kernel, according to [25]. We also tried the same pipeline, but replacing the fast SIFT with dense Daisy features[26].

All the methods were implemented in C/C++, using the original author's code when possible. They were chosen for comparison as each of them was developed with the aim of obtaining a good balance of speed and accuracy. Multiple working points were tested for each of these methods, representing various optimization for speed and accuracy. For the fast SIFT method, shifting between speed and accuracy was done by changing the stride parameter, controlling the density of the SIFT greed. For the Daisy we also choose the Daisy complexity to optimize speed/accuracy, as recommended in [26].

The (CPU time, accuracy) of the best working points obtained by each of the algorithms, including DFE, are plotted together in Figure 4 (left). We see that random forest can achieve similar classification time to that of DFE, but is significantly less accurate (FN=10.6% vs. FN=2% for DFE, for the same CPU budget). Consistent with [15], we found that the best accuracy is achieved by training on a small number of deep trees, with little improvement when this number is increased. This leaves us with less flexibility on controlling the tradeoff between accuracy and classification time. There are several reasons why using 50 ferns DFE is about as fast as using two trees. First, each fern operates on relatively small number of pixels (50), which is only ˜4% of the image. Second, calculating the ferns bits requires less operations than forest with the same depth, as discussed in Section 3.2. Third, the number of bits per fern is 13, while the depth of tree is 21. Also, the memory size of the forest is in order of 80MB vs. 2.5MB of ferns. Since 80MB cannot fit into the cache, we pay with more cache misses.

The accuracy of with fast SIFT and Daisy alternatives, can approach the accuracy of the DFE. However, their classification time is two order of magnitudes longer. By optimize them for speed we significantly loose accuracy without getting to the target classification time.

In the next section, we show that in addition to high accuracy and fast classification, DFE approach enable significant flexibility for various trade-offs of speed, accuracy, memory size and generalization from various sizes of training set.

4.5. Training Sample Size and Memory

As discussed before, the fern ensemble architecture trades speed and accuracy for sample size and memory. For each training set size, constraints on memory and classification time we optimize accuracy by tuning M and K. In this section we show that increasing the training set size enable us not only to improve accuracy, but also to significantly reduce the classification time.

Figure 4 (middle) shows the effect of increasing the training set size on FN, for fixed M and K. We modify the training set size we use from ˜0.2% of the full set (820 images) to the full training set (420,000 images). The subset of training set is selected randomly. As expected, the false negative rate reduces with increase of training set size.

In our problem, however, even with a training set size of ˜30,000 samples (0.07 in x-axis of Figure 4) the accuracy we got met minimum requirements for the product. However, even after full code optimization, the classification time significantly exceeded the target budget. The question is if we can reduce classification time by increasing the training set size and modifying M and K.

Figure 4 (right) shows the classification time as a function of the of training set size, relative to the full set, for various target false negative rates. We can see that for a fixed target accuracy, the classification time can be reduced by an order of magnitude, if we increase the training set size by an order of magnitude. In general, as training set size increases, we slightly increase K and significantly reduce M to achieve same target accuracy with lower classification time. This can be explained by the effect of K on the capacity of each fern, and hence should be adapted to

LUT entries	Ferns # (M)	Bits # (K)	% FN @ FP=2%
768	48	4	10.7
1536	96	4	7.78
3072	96	5	6.07
6144	192	5	5.42
12288	384	5	4.21
24576	384	6	2.97
49152	768	6	2.32

Table 1. DFE accuracy under memory limits. LUT entries is the total number of entries in all the lookup tables (ferns) together, which is $2^K M$. In our implementation, each LUT entry requires 6 bytes - two bytes per class, representing the SVM weights.

the training set size. On the other hands, the accuracy can be improved by increasing M, but at a significant cost of classification time. These results are significant for building practical systems. While it is well know that increasing training set size enables improvement in accuracy, here we show that it can also reduce classification time significantly.

Another interesting tradeoff enabled in the DFE framework is between memory and accuracy. Table 1 presents false negative rate versus memory consumption for a fern ensemble. Memory consumption can be reduced by lowering either M or K, and in the table we chose the optimal M, K parameters for each memory limit point. From the table we can see adding a memory constraint leads to significant reduction in the number of bits per fern, and increasing the number of ferns. The result is very different from the case of optimizing for classification time, where optimal number of bits is high. This is not surprising, as the memory size increases exponentially with number of bits, but classification time increases only linearly. The result classification time is about 5-10 larger when we optimize for memory instead of for speed. Note, however, that in our baseline implementation, with 50 ferns and 13 bits the memory size is about 2.5MB, which still fits into the cache.

4.6. MNIST result

We tested the relevance of the DFE classifier to non-depth imagery on the MNIST digit dataset, containing $60,000$ training images of size 28×28 from 10 digit classes, and $10,000$ test images. A DFE with $M = 10$ ferns and $K = 13$ bits was trained, with quadrant ferns and no checkerboard. The DFE was used as it was used for hand pose, without any domain adaptation or pre-processing of the digit images. The test error was 0.77%, which is among the best results obtained without the usage of virtual samples. The classifier uses only 8320 pixel comparisons and runs in 0.011 millisecond, compared to $2 - 20$ millions of floating point operations in the leading CNN methods.

5. Conclusions and Further Work

We have seen that the *discriminative fern ensemble* framework enables a significant push of the accuracy-speed envelope for visual recognition. Thin, efficient architecture, and discriminative optimization were found important for this purpose. In terms of architecture, it would be interesting to extend the table-based approach to deeper models with more table layers. Another interesting direction is to analyze the trade-off between classification time and training sample size for other algorithms.

References

[1] A. Bar-Hillel, D. Hanukaev, and D. Levi. Fusing visual and range imaging for object class recognition. In *ICCV*, 2011. 2

[2] A. Bar-Hillel, D. Levi, E. Krupka, and C. Goldberg. Part-based feature synthesis for human detection. In *ECCV*, 2010. 1, 3.3, 3.3, 3.3

[3] R. Benenson, M. Mathias, R. Timofte, and L. J. V. Gool. Pedestrian detection at 100 frames per second. In *CVPR*, 2012. 1

[4] J. Bi, T. Zhang, and K. Bennett. Column-generation boosting methods for mixture of kernels. In *KDD*, 2004. 3.3

[5] L. Bo, K. Lai, X. Ren, and D. Fox. Object recognition with hierarchical kernel descriptors. In *CVPR*, 2011. 2

[6] A. Bosch, A. Zisserman, and X. Muñoz. Image classification using random forests and ferns. In *ICCV*, pages 1–8, 2007. 1, 1, 2, 4.3

[7] M. Chen, Z. E. Xu, K. Q. Weinberger, O. Chapelle, and D. Kedem. Classifier cascade for minimizing feature evaluation cost. In *AISTATS*, 2012. 2

[8] T. Dean, M. Ruzon, M. Segal, J. Shlens, S. Vijayanarasimhan, and J. Yagnik. Fast, accurate detection of 100,000 object classes on a single machine. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, Washington, DC, USA, 2013. 1

[9] T. G. Dietterich and D. Fisher. An experimental comparison of three methods for constructing ensembles of decision trees. In *Machine Learning*, pages 139–157, 2000. 2

[10] P. Doliotis, V. Athitsos, D. I. Kosmopoulos, and S. J. Perantonis. Hand shape and 3d pose estimation using depth data from a single cluttered frame. In *ISVC*, 2012. 2

[11] P. Dollár, S. Belongie, and P. Perona. The fastest pedestrian detector in the west. In *BMVC*, 2010. 1

[12] P. Dollár, Z. Tu, P. Perona, and S. Belongie. Integral channel features. In *BMVC*, 2009. 2

[13] A. Erol, G. Bebis, M. Nicolescu, R. D. Boyle, and X. Twombly. Vision-based hand pose estimation: A review. *Comput. Vis. Image Underst.*, 108(1-2):52–73, 2007. 2

[14] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008. 4.2

[15] C. Keskin, F. Kirac, Y. E. Kara, and L. Akarun. Hand pose estimation and hand shape classification using multi-layered randomized decision forests. In *ECCV*, pages 852–863, 2012. 1, 2, 4.4, 4

[16] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, pages 2169–2178, 2006. 1

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, 1998. 4

[18] D. Levi, S. Silberstein, and A. Bar-Hillel. Fast multiple-part based object detection using kd-ferns. In *CVPR*, 2013. 1, 1, 2

[19] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004. 1

[20] F. Moosmann, B. Triggs, and F. Jurie. Fast discriminative visual codebooks using randomized clustering forests. In *NIPS*, 2007. 1, 4.4

[21] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua. Fast keypoint recognition using random ferns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(3):448–461, 2010. 1, 2, 3.1, 3.3, 4.3

[22] J. Shotton, T. Sharp, A. Kipman, A. W. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore. Real-time human pose recognition in parts from single depth images. *Commun. ACM*, 56(1):116–124, 2013. 1, 1, 2, 4.1

[23] E. Tola, V.Lepetit, and P. Fua. A Fast Local Descriptor for Dense Matching. In *CVPR*, 2008. 1

[24] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. http://www.vlfeat.org/, 2008. 4.4

[25] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. *Pattern Analysis and Machine Intellingence*, 34(3), 2011. 1, 4.4

[26] S. Winder, G. Hua, and M. Brown. Picking the best daisy. *CVPR*, 2009. 1, 4.4