MITIGATING MEMORY REQUIREMENTS FOR RANDOM TREES/FERNS

*C. De Vleeschouwer, A. Legrand, L. Jacques**

Université catholique de Louvain
ICTEAM Institute, Belgium

Martial Hebert

Carnegie Mellon University
The Robotics Institute, USA

ABSTRACT

Randomized sets of binary tests have appeared to be quite effective in solving a variety of image processing and vision problems. The exponential growth of their memory usage with the size of the sets however hampers their implementation on the memory-constrained hardware generally available on low-power embedded systems. Our paper addresses this limitation by formulating the conventional semi-naive Bayesian ensemble decision rule in terms of posterior class probabilities, instead of class conditional distributions of binary tests realizations. Subsequent clustering of the posterior class distributions computed at training allows for sharp reduction of large binary tests sets memory footprint, while preserving their high accuracy. Our validation considers a smart metering applicative scenario, and demonstrates that up to 80% of the memory usage can be saved, at constant accuracy.

***Index Terms*—** Random ferns, random trees, memory usage.

I. INTRODUCTION AND POSITIONING

In the last decade, random forests have emerged as effective and versatile ensemble-based decision systems [1], [2], [3], [4]. Their versatility arises from the arbitrary definition of application-dependent attributes to support the binary partition in the tree nodes [3], [5], but also from the fact that the leaves of the trees can be mapped to an arbitrary space, including a continuous space for regression [6], [7], a topologically structured label space for label correlation exploitation [5], or even a Hough space to detect specific configurations of a parametric signal [8]. The accuracy and the good generalization properties of random forests have been demonstrated in a broad variety of applications and scenarios, including image classification [9], [10], [11], [12], pose, action, and defect recognition [13], [14], [15], keypoints matching [16], [17], or image enhancement [18], to name a few. Our paper primarily investigates how random forests can be embedded in a resource-constraint hardware platform. In this introduction, we first review the main factors affecting the forests hardware resources. We then survey some works related to their efficient implementation, before presenting our contribution and the outline of the paper.

I-A. Random forest resources

This paragraph reviews a number of factors affecting the computational and memory resources of a random forest.

A first one is certainly the number of tests, which depends both on the depth and the number of tree models. Since the trees are trained and evaluated independently, both the memory and computational resources increase linearly with the number of trees in the forest. In contrast, increasing the tree depth by one doubles the number of leaves and associated parameters, but has negligible effect on the run-time speed [19]. In terms of decision accuracy, previous works have shown that the ensemble performances tend to saturate beyond a sufficient number of tree structures, typically beyond (a few) hundred(s) of trees [20], [19], while increasing the depth is known to significantly increase the accuracy [10], [14], as long as enough training samples are available to estimate the increased number of parameters. Hence, deeper trees are recommended since they can handle more variations in the image at hand, without much of a slow-down. This decision accuracy improvement is however obtained at the cost of additional training resources, and of an exponential growth of the number of leaves, and associated memory resources.

A second important factor affecting the random trees resources consists in the strategy adopted to turn the multiple trees outcomes into a single and joint decision, since it directly impacts the amount of parameters and thus the memory associated to each leave of the trees. In the training phase, each tree evaluates the training input samples, and learns either the majority class or the class posterior distributions in each one of its leaves. During testing, either majority voting or class posterior distributions averaging has long been used to infer the ensemble decision from the trees outputs [2], [9], [10]. A Semi-Naive Bayesian inference has recently been demonstrated to significantly outperform those approaches[19], based on the product of class conditional probabilities. In our experiments, we naturally rely on this Semi-Naive Bayesian formulation to combine the outcomes of the trees. Not using a majority vote strategy means that a class conditional distribution, whose memory footprint increases exponentially with the depth of the tree, should be stored for each class.

A third and last factor impacting the complexity lies in the strategy adopted to define the tests in the nodes of the ensemble of trees. In his seminal work [2], Breiman recommends bagging and random feature selection at each node to construct a collection of diverse whilst accurate decision trees. Most recent works have abandoned bagging, but still randomize both the attributes and the cut-point choices to define the binary test in a tree node. Typically, entropy-based criterions are considered to select the binary test that provides the highest information gain, among R randomly chosen tests. In the extreme case, *i.e.*, when $R = 1$, totally randomized trees are built. Extreme randomization offers fast training compared to other ensemble methods, and has been shown to be remarkably accurate in a variety of tasks, *e.g.*, [9], [10], as long as a reasonable amount of structures are considered in the ensemble of decision structures [3]. Extreme randomization builds structures that are independent of the learning samples, and makes thus the selection of a child node binary test independent of the tests adopted for its parents. This obviously questions the relevance of the hierarchical tree structure. This point has been investigated in [19], where the authors observe that replacing the hierarchical structure of the tree by a flat one does not affect the decision accuracy. Those flat structures, named ferns, offer data independent processing time, and more regular memory access patterns. Hence, they generally lead to a runtime improvement, while mostly preserving the decision accuracy [21]. For this reason, whilst our contributions remain valid for hierarchical trees, most of the descriptions and validations provided in our paper deal with random ferns.

As a conclusion, despite their numerous advantages, a fundamental drawback of random trees/ferns lies in their exponential

*Part of this work has been funded by the Belgian N.S.F., and by the Walloon Region Mecatech project SAVE

growth of memory usage with the depth of the trees [22]. This exponential growth poses a problem for implementing high-accuracy tree models on memory-constrained hardware such as embedded processors, because (i) deep trees or large ferns are required to achieve high-accuracy, and (ii) storage of the class conditional distributions of tree leaves or fern values is required to implement accurate ensemble inference strategy.

I-B. Related works

Several previous works have attempted to mitigate the implementation issues raised by random forests.

In [23], the authors implement decision trees on a GPU. They first map the data structure describing a decision forest to a 2D texture array, and then evaluate the tree decisions and compute the leaves histograms using a combination of pixel shaders and vertex shaders. They do not present any new theory but concentrate on leveraging the full parallelism of the GPU, which yields a two orders of magnitude performance increase over a standard CPU implementation. However, their work relies on strong parallelisation and large size memory, and is thus not suited to embedded systems.

The work in [24] points out the fact that applying hardware acceleration to random forests is challenging when the decision trees vary significantly in terms of shape and depth. This is because the irregularities in tree sizes and structures complicate the memory accesses, and make the time to process a sample data dependent. The authors also observe that a random forest is a real memory guzzler: the limited memory ressources of the FPGA constrain the trees depth. As a consequence, they recommend the use of *compact random forests*, composed of many, small trees rather than fewer, deep trees. As explained above, this penalizes the decision accuracy.

To limit the memory consumption, [25] introduces the notion of decision jungles, by extending the tree structure to a rooted directed acyclic graph (DAG). The main advantage of DAG is the capacity to bound memory consumption, by specifying a maximal width at each layer in the DAG. Their practical implementation requires the use of learning algorithms that, within each level, jointly optimize an objective function over both the structure of the graph and the nodes binary tests features. At constant memory usage, the DAG structure has been shown to improve generalization compared to traditional trees when the number of structures is reduced [25], but no gain in accuracy has been reported when the number of structures increases at levels recommended in [3], [19].

I-C. Contribution and outline

In this paper, we present a solution to mitigate the explosion of memory requirements induced by high-accuracy ensembles of trees/ferns. We first express the conventional semi-naive Bayesian ensemble decision rule in terms of posterior class probabilities, instead of class conditional distributions. The main idea of our contribution then consists in clustering the class posteriors distributions learned for each leave of a tree or, equivalently, for each value of a fern, so that each class posterior can be approximated by its cluster center. Hence, despite the number of class posteriors increases exponentially with the size of the fern or the depth of the tree, we can limit the memory footprint to one distribution per cluster. Our experiments reveal that the resulting approximation of the class posteriors does not cause a sharp decrease of the ensemble accuracy. They also demonstrate that significantly improved memory/accuracy trade-offs are achieved when combining larger ferns (or deeper trees) with class posteriors probabilities clustering.

The paper is organized as follows. Section II reviews the random ferns principle, and quantifies their memory requirements. Section III presents our contribution, which restricts memory requirements to the storage of class posteriors centroids distributions, in addition to a compact look-up table assigning a centroid index to each fern value (tree leave). Section IV validates our approach in terms of memory/accuracy trade-offs, in the context of a low-power smart metering system. It demonstrates the advantage of clustering the posteriors globally, i.e. jointly for all ferns, compared to defining clusters on a fern basis. Section V concludes.

II. MEMORY USAGE IN FERNS/TREES

To motivate our contribution, it is important to quantify the memory consumption associated to an ensemble of random classifiers. Without loss of generality, we restrict our developments to random ferns (RF) [19], and show that memory consumption is dominated by the class conditional distributions of ferns.

II-A. Ferns definition

This section reviews the principle underlying the classification with an ensemble of random sets of binary tests. The approach considers $N \in \mathbb{N}$ sets of $S \in \mathbb{N}$ binary tests that are randomly selected, to define N flat structures, named *ferns*. Generalization to hierarchical tree structures is straightforward. Hence, in the rest of the paper, we only consider the fern paradigm.

As in [19], let $C \in \mathcal{C}$ denote the random variable that represents the class of an image sample, and $\mathcal{C} = \{c_i : 0 < i \leq H\}$ be the set of $H \in \mathbb{N}$ classes c_i. Given the ensemble of N ferns $\mathcal{F} = \{F_k \in \{0,1\}^S : 1 \leq k \leq N\}$, the sample class MAP estimate $\hat{c}$ is defined by:

$$\hat{c} = \arg\max_{c_i} P(C = c_i | F_1, \cdots, F_N). \tag{1}$$

If we admit a uniform prior $P(C = c_i) = 1/H$, Bayes' formula yields:

$$\hat{c} = \arg\max_{c_i} P(F_1, \cdots, F_N | C = c_i). \tag{2}$$

Learning and handling the joint probability in (2) is not feasible for large $N \times S$ products since it would require to compute and store 2^{NS} entries for each class. To keep the conditional probabilities tractable while accounting for some binary tests dependencies, the semi-naive Bayesian approach proposed in [19] assumes independence between the ferns, but accounts for dependencies between the binary tests belonging to the same fern. The joint conditional probability is approximated by:

$$P(F_1, \cdots, F_N | C = c_i) = \prod_{k=1}^{N} P(F_k | C = c_i), \tag{3}$$

where F_k denotes the k^{th} fern, and the class conditional distribution of each fern is simply learnt based on the accumulation of the observations, as detailed in [19].

II-B. Memory consumption

In terms of memory, Random Ferns have to store (i) the list of binary tests composing each fern, and (ii) the class conditional distributions of all ferns.

In practical cases, the memory required to store the conditional distributions is by far larger than the one needed for the binary tests parameters. This is especially true when considering large size ferns, since the amount of tests parameters increases proportionally to the fern size S while, in contrast, the dimension of the class conditional distributions grows exponentially with S. Specifically, letting B denote the number of bits required to quantize the probability values, the class conditional distributions memory requirement for each fern becomes $B \times H \times 2^S$. As an illustrative example, in the metering use case considered in Section IV, $H = 10$, B is set to 8, and S ranges between 6 (low classification accuracy) and 10 (high classification accuracy). It means that the class conditional distribution memory consumption ranges between 5 and 82 kbits per fern. In comparison, assuming that the application works on normalized 32×16 digit images, each binary test spends 18 bits to define the position of the pixels

involved in the test, which means between 108 ($S = 6$) and 180 ($S = 10$) bits per fern. In absolute terms, 82 kbits/ferns means more than 1 MBytes for 100 ferns. This requirement obviously increases with S, H and B, and is prohibitive for low-power memory constrained embedded systems[1].

This tends to encourage the use of small ferns. This conclusion is in line with the literature since, under strict memory constraints, [19] has shown that the best classification rates are obtained by using many small ferns, mainly because -under memory constraint- the number of ferns sharply decreases when their size increases. However, several works have shown that using small size ferns penalizes the classification decision since the accuracy tends to saturate beyond a sufficient number of ferns (see Fig. 7 in [19]), at a level that decreases with the size of the structure (see Fig. 4 in [10], or Fig. 2(b) in our validation). Hence, as long as there are enough of them, large ferns are more accurate than shorter ones. This is mainly because they are able to capture a larger variety of class-specific image configurations, by exploiting the dependencies between a larger number of binary tests. We conclude that the memory constraint prevents the implementation of high-accuracy random ferns on low-power embedded platforms.

III. POSTERIOR DISTRIBUTIONS CLUSTERING

The contribution presented in this section fundamentally aims at mitigating the memory consumption when handling large ferns (or, equivalently, deep trees). It formulates the random ferns decision rule in terms of class posteriors (instead of class conditionals), and then aggregates those class posteriors distributions into a limited number of clusters to reduce the storage requirements.

III-A. Class posteriors clusters

Under the same uniform prior assumption than the one adopted to derive (2), we use Bayes' formula to develop each factor in (3), and find

$$\hat{c} = \arg\max_{c_i} \prod_{k=1}^{N} P(C = c_i|F_k). \tag{4}$$

Hence, handling the class posteriors $P(C = c_i|F_k)$ as in (3) is equivalent to maintaining and manipulating the class conditional distributions $P(F_k|C = c_i)$. Despite the storage requirement is obviously the same for both sets of distributions, this observation turns the storage of H vectors of dimension 2^S into a storage of 2^S vectors of dimension H. When the fern size S increases, the class posteriors define a very large set of constant-dimension probability distributions, while the class conditional distributions increase exponentially in dimension. Using class posteriors replaces thus the explosion of distribution dimension by an explosion of the number of distributions.

Among the huge set of class posterior distributions resulting from the combination of random binary partitioning, one might expect to observe similar vectors, reflecting a limited number of class posterior trends. This suggests to group the class posteriors into a limited number of clusters K, and to approximate each class posterior by the mean of its cluster, *i.e.*, we reduce the dimensionality of the vectors set. In that way, only the K mean distributions need to be stored, in addition to the table assigning each fern value to its associated cluster. In practice, the clusters can be computed either on a fern basis, or globally for all ferns. In the following, we use the variable K to denote the number of clusters when they are computed independently on each fern, and K_g when we refer to the clusters that are computed globally, to approximate the posteriors observed for all ferns.

The clustering reduces the above $N \times [B \times H \times 2^S]$ class posteriors memory footprint to either $N \times [K \times B \times H + 2^S \times log_2(K)]$ or $K_g \times B \times H + N \times 2^S \times log_2(K_g)$, depending on whether we consider fern-based or global clustering, respectively. Here, we assume that the cluster indices are encoded with constant-length codewords of length $log_2(K)$ or $log_2(K_g)$. This is an upper bound since some clusters might be more popular than others, thereby reducing the actual entropy of the cluster indice variable.

From those formulas, we observe that the clustering strategy, either global or fern-based, makes the exponential growth of memory induced by the increase of S (see the 2^S factor) independent of the number of class H. This makes it especially appealing for problems with large number of classes or when the output space has to approximate a continuous space.

III-B. K-means clustering

Formally, for a given fern, let $\boldsymbol{v}_j \in \mathbb{R}_+^H$ denote the distribution associated to the j^{th} fern value, *i.e.*, $(\boldsymbol{v}_j)_i = P(C = c_i|F = f_j)$. As told in Section II-A, this vector is derived from the accumulation of the observations of the fern realizations over the set of training samples. Specifically, let n_j^i be the number of training samples of class c_i that evaluate to the j^{th} fern value f_j. We also define $n_j = \sum_i n_j^i$, to denote the number of training samples that evaluate to f_j. To prevent issues related to the fact that no training sample of one class might evaluate to some fern value, similarly to [19], we introduce a regularization term $R = 0.1$ while estimating the probability distribution $\boldsymbol{v}_j$. Hence, we have

$$(\boldsymbol{v}_j)_i = \frac{n_j^i + R}{n_j + H \times R}, \tag{5}$$

with H denoting the number of classes. From the training stage, we also record an estimate of the frequency w_j at which each fern value f_j is observed among the training samples, and write:

$$w_j = \frac{n_j + H \times R}{\sum_i (n_i + H \times R)}. \tag{6}$$

Consider now the problem of clustering the 2^S distributions $\boldsymbol{v}_j$ into K clusters with $K < 2^S$. Assuming that the set $\{\boldsymbol{m}_k \in \mathbb{R}_+^H : 1 \leq k \leq K\}$ denotes the K initial cluster centroids, the K-means algorithm iterates between the following two steps [26]:
(i) Centroid assignment. Given the ℓ_2-distance $d : \mathbb{R}_+^H \times R_+^H \rightarrow R_+$, we form the class $\mathcal{M}_k = \{\boldsymbol{v}_j : d(\boldsymbol{v}_j, \boldsymbol{m}_k) \leq d(\boldsymbol{v}_j, \boldsymbol{m}_{k'}), \forall k' \neq k\}$.
(ii) Centroid update. We update the centroid as the average of the class posteriors assigned to the cluster, weighted by their respective occurence frequencies w_j, so as to minimize the expected mean squared error on the training set, *i.e.*,

$$\boldsymbol{m}_k \leftarrow (|\mathcal{M}_k|)^{-1} \sum_{v_i \in M_k} w_i . \boldsymbol{v}_i,$$

where $|A|$ is the cardinality of a set A.

Alternating between these two steps until convergence minimizes the squared approximation error of the set reduction. Notice that the initialization is not a critical component of our clustering problem. Indeed, since the clustering is only considered during the training stage, it has no impact on the run-time performances. Hence, one can simply run the clustering multiple times, starting with different random initialization seeds, and select the convergence point that minimizes the approximation error.

When the class posterior distributions of each fern are clustered independently, the above processing is repeated N times, to define $N \times K$ centroids.

As an alternative, we have also considered the global joint clustering of the whole set of distributions associated to the N ferns. The extension of the fern-based clustering to a global one is trivial, and simply consists in clustering the $N \times 2^S$ distributions associated to the N ferns into K_g clusters. As confirmed by our experiments, computing a single set of class posteriors centroids for the whole set of ferns is able to save even more memory compared to using N distinct set of centroids (one for each fern).

[1]Off-the-shelf low power ARM Cortex M3 or M4 embed memories ranging from 16 to 180 kB, see STM32L1 or ATSAM4 series.

IV. VALIDATION

IV-A. Use case

We consider the optical recognition of digits with an ultra low-power vision system. The ultra low power constraint arises from the fact that the system has to run for years on a battery, without recharging. Such applicative scenario is for exemple relevant for easy deployment of smart metering systems, to monitor water/gas/electricity consumption from remote meter reading.

Random ferns are envisioned as an alternative to conventional SVM approaches in this context because, despite their excellent performances, (i) polynomial kernel SVMs trained on raw pixels are known to be impractical as they have a huge complexity at runtime [27], and (ii) linear SVM require the extraction of appropriate features to compete with state-of-the-art in terms of accuracy [27], [28], [29], which implies some additional data processing (and thus some energy). In contrast, random ferns only require $N \times S$ raw pixels comparisons, and $(N-1) \times H$ additions when the products of probabilities are implemented based on sums of logarithms. This means about 20 times less operations than what is required by the most efficient linear SVM solution, proposed in [27], based on histograms-of-gradients. The gain in instructions numbers will be even higher on most platforms since random ferns only require pixel value comparisons where SVMs need multiply-add operations.

This gain in number of instructions largely impacts the system power consumption, since data processing tends to consume more energy than image capture on modern ultra-low power vision systems. Typically, a state-of-the-art ultra-low power image sensors consumes as little as 17 pJ/pixel, which means about 0.5 μJ for a 30 kpixels image [30]. In contrast, ultra-low power processors require 8-15 pJ/instruction [31], [32], which means around 1 μJ to recognize the ten digits in a typical meter image with our ferns, assuming that each operation corresponds to 4 to 5 instructions. Those figures are based on a relatively small RAM memory (32 kB for program + 32 kB for data), and are expected to increase with larger memory sizes. They however demonstrate the relevance of reducing data processing needs, and motivate solutions that can live with less than one hundred kB of embedded memory, i.e. below the upper limit offered by off-the-shelf low-power microcontrollers. The experiments below show that our proposed class posterior distributions clustering effectively reduces memory requirements to those ranges, while preserving decision accuracy.

IV-B. Results

Our application assumes that the bounding boxes surrounding the digits to recognize are defined through prior calibration. Hence, the observed digits are only affected by distortions that are related to the placement of the camera (projective distortion), to the calibration of the system (rotation, scaling, and shifting of the digit inside the bounding box), and to the quality of the acquisition itself (blurring and noise due to lightning). In our validation, without loss of relevance w.r.t. our memory usage question, we consider a synthetic dataset, emulating the expected distortions by transforming digits printed in a 42×24 image. Transformations include rotations by angles $\in [-6°, +6°]$, scaling by factors $\in [0.8, 1.2]$, horizontal and vertical shifts by vectors $\in [-4, 4] \times [-4, 4]$, blurring by a Gaussian filter with $\sigma \in [0.5, 2]$, and projective transforms (approximated by moving independently the corners of the digit image by vectors $\in [-5, 5] \times [-5, 5]$). Examples of distorted images are provides in Figure 1.

Fig. 1: Dataset image samples, obtained by combining multiple distortions.

We have run a ten-fold cross validation to measure the accuracy obtained for different random ferns parameters B, N, S, and K.

In Figure 2, we consider the conventional random ferns algorithm. On the left, Figure 2(a) presents the accuracy as a function of the number of bits B used to quantize the probability values. We observe that the accuracy tends to saturate beyond B=8. Hence, B has been set to 8 in all our remaining experiments. On the right, Figure 2(b) confirms that increasing the number of structures in the ensemble is not sufficient to achieve high decision accuracies. As expected, high-accuracy can only be obtained by increasing the size of the ferns. This confirms the relevance of our study since large ferns size means huge memory usage (exponential growth).

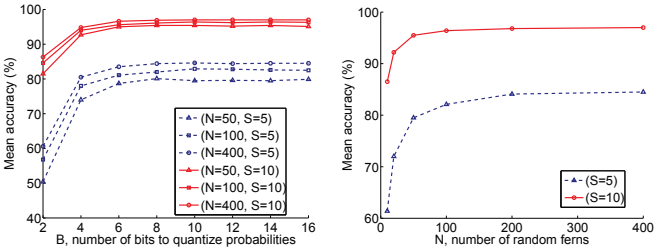

Fig. 2: Accuracy of conventional random ferns as a function of quantization depth B (a), and number of ferns N (b), for different fern sizes S.

Figure 3 compares the accuracy/memory trade-offs obtained with the conventional algorithm (blue and red solid lines, with $K = 2^S$) to the one obtained based on our proposed clusterization strategy (red dashed lines for K fern-based clusters, and green solid line for K_g global clusters). We observe that the global clustering strategy achieves the best accuracy/memory trade-offs. It achieves up to 5% higher accuracy at constant memory usage, compared to conventional random ferns. It also significantly reduces memory usage at constant accuracy. For high-accuracy models (see Figure 3(b)), it can save as much as 80% of memory usage, at constant accuracy.

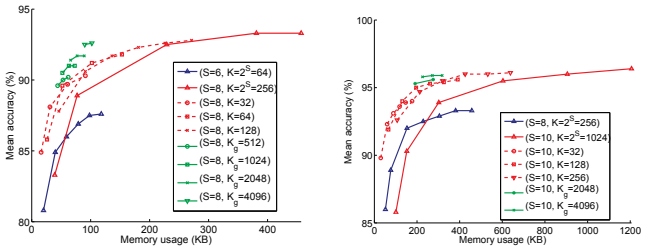

Fig. 3: Accuracy as a function of the total (tests+class posteriors) memory usage, for different fern size S and number of clusters. K refers to the number of fern-based clusters, while K_g denotes the number of global clusters, computed for the whole set of ferns. Red and blue solid lines denote the standard algorithm.

V. CONCLUSION

This paper has turned the class conditionals decision rule generally adopted for random ferns/trees into an equivalent class posteriors rule. It has then proposed to cluster the class posterior probabilities associated to the realisation of a set of weak binary tests, so as to implement ensemble of randomized trees or ferns classifier under severe memory constraints. The approach appears to be both flexible and effective, and unbolt the lock for implementing high-accuracy tree models on memory-constrained hardware such as embedded or mobile processors.

VI. REFERENCES

[1] Y. Amit and D. Geman, "Shape quantization and recognition with randomized trees," *Neural Computation*, vol. 9, no. 12, pp. 1545–1588, 1997.

[2] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[3] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely Randomized Trees," *Machine Learning*, vol. 36, no. 1, pp. 3–42, 2006.

[4] A. Criminisi and J. Shotton, "Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning," *Foundations and Trends in Computer Graphics and Computer Vision*, vol. 7, no. 2-3, pp. 81–227, February 2012.

[5] P. Kontschieder, S.R. Bul, H. Bischof, and Pelillo M., "Structured class-labels in random forests for semantic image labeling," in *ICCV*, 2011.

[6] A. Montillo and Haibin Ling, "Age regression from faces using random forests," in *ICIP*, October 2009.

[7] B. Glocker, O. Pauly, E. Konukoglu, and A. Criminisi, "Joint classification-regression forests for spatially structured multi-object segmentation," in *ECCV*, October 2012.

[8] J. Gall, A. Yao, N. Razavi, L. Van Gool, and V. Lempitsky, "Hough forests for object detection, tracking, and action recognition," *IEEE Trans. on PAMI*, vol. 33, no. 11, pp. 2188–2202, 2011.

[9] R. Marée, P. Geurts, J. Piater, and L. Wehenkel, "Random subwindows for robust image classification," in *IEEE CVPR*, 2005, pp. 34–40.

[10] A. Bosch, A. Zisserman, and X. Munoz, "Image classification using random forests and ferns," in *ICCV*, 2007.

[11] Bingbing Ni, Shuicheng Yan, Meng Wang, A.A. Kassim, and Qi Tian, "High-order local spatial context modeling by spatialized random forest," *IEEE Trans. on Image Processing*, vol. 22, no. 2, pp. 739 – 751, 2013.

[12] M. Torres and Guoping Qiu, "Habitat classification using random forest based image annotation," in *ICIP*, September 2013.

[13] G. Rogez, J. Rihan, S. Ramalingam, C. Orrite, and P. H. S. Torr, "Randomized trees for human pose detection," in *IEEE CVPR*.

[14] 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 *IEEE CVPR*.

[15] Fei Zhao, P.R.S. Mendonca, Jie Yu Yu, and R. Kaucic, "Learning-based automatic defect recognition with computed tomographic imaging," in *ICIP*, September 2013.

[16] V. Lepetit and P. Fua, "Keypoint recognition using randomized trees," *IEEE Trans. on PAMI*, vol. 28, no. 9, pp. 14651479, 2006.

[17] Zhi Zhou, Yue Wang, and Eam Khwang Teoh, "Robust object tracking using bi-model," in *ICIP*, September 2013.

[18] D. Alexander, D. Zikic, J. Zhang, H. Zhang, and A. Criminisi, "Image quality transfer via random forest regression: Applications in diffusion mri," in *MICCAI*. Springer, 2014, vol. 17, pp. 225–232.

[19] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua, "Fast keypoint recognition using random ferns," *IEEE Trans. on PAMI*, vol. 32, no. 3, pp. 448–461, 2010.

[20] A. Criminisi and J. Shotton, "Decision forests for computer vision and medical image analysis," *Advances in Computer Vision and Pattern Recognition, Springer*, vol. 19, 2013, ISBN 978-1-4471-4929-3.

[21] M. Donoser and D. Schmalstieg, "Discriminative feature-to-point matching in image-based localization," in *IEEE CVPR*.

[22] V. Lepetit and P. Fua, *Keypoint Recognition Using Random Forests and Random Ferns*, chapter 9, pp. 111–124, Springer, 2013.

[23] Sharp T., "Implementing decision trees and forests on a gpu," in *ECCV*, 2008.

[24] B.C. Van Essen, C.C. Macaraeg, R. Prenger, and M. Gokhale, "Accelerating a random forest classifier: multi-core, GP-GPU, or FPGA?," in *IEEE International Symposium on Field-Programmable Custom Computing Machines*.

[25] J. Shotton, T. Sharp, P. Kohli, S. Nowozin, J. Winn, and A. Criminisi, "Decision jungles: Compact and rich models for classification," in *Neural Information Processing Systems Foundation*.

[26] J. Han and Kamber M., *Cluster Analysis*, chapter 7, pp. 402–406, Elsevier, 2006.

[27] S. Maji, A.C. Berg, and J. Malik, "Efficient classification for additive kernel svms," *IEEE Trans. on PAMI*, vol. 35, no. 1, pp. 66–77, 2013.

[28] D. Delannay, N. Danhier, and C. De Vleeschouwer, "Detection and recognition of sports (wo)men from multiple views," in *3rd ACM/IEEE International Conference on Distributed Smart Cameras*, 2009, pp. 1–7.

[29] C. Verleysen and C. De Vleeschouwer, "Recognition of sport players' numbers using fast-color segmentation," in *Proc. SPIE 8305, Visual Information Processing and Communication III*, February.

[30] D. Bol, G. de Streel, F. Botman, A. Kuti Lusala, and N. Couniot, "A 65-nm 0.5-v 17-pj/frame.pixel dps cmos image sensor for ultra-low-power socs achieving 40-db dynamic range," in *IEEE Symposia on VLSI Technology and Circuits*, June 2014.

[31] F. Botman, J. De Vos, S. Bernard, F. Stas, J.D. Legat, and D. Bol, "Bellevue: a 50mhz variable-width simd 32bit microcontroller at 0.37v for processing-intensive wireless sensor nodes," in *IEEE International Symposium on Circuits and Systems*, June 2014.

[32] D. Bol, J. De Vos, C. Hocquet, F. Botman, F. Durvaux, S. Boyd, D. Flandre, and J.D. Legat, "A 25mhz 7w/mhz ultra-low-voltage microcontroller soc in 65nm lp/gp cmos for low-carbon wireless sensor nodes," in *IEEE International Solid-State Circuits Conference*, February 2012.