On-line Random Forests

Amir Saffari Christian Leistner Jakob Santner Martin Godec Horst Bischof
Institute for Computer Graphics and Vision
Graz University of Technology
{saffari,leistner,santner,godec,bischof}@icg.tugraz.at

Abstract

Random Forests (RFs) are frequently used in many computer vision and machine learning applications. Their popularity is mainly driven by their high computational efficiency during both training and evaluation while achieving state-of-the-art results. However, in most applications RFs are used off-line. This limits their usability for many practical problems, for instance, when training data arrives sequentially or the underlying distribution is continuously changing.

In this paper, we propose a novel on-line random forest algorithm. We combine ideas from on-line bagging, extremely randomized forests and propose an on-line decision tree growing procedure. Additionally, we add a temporal weighting scheme for adaptively discarding some trees based on their out-of-bag-error in given time intervals and consequently growing of new trees. The experiments on common machine learning data sets show that our algorithm converges to the performance of the off-line RF. Additionally, we conduct experiments for visual tracking, where we demonstrate real-time state-of-the-art performance on well-known scenarios and show good performance in case of occlusions and appearance changes where we outperform trackers based on on-line boosting. Finally, we demonstrate the usability of on-line RFs on the task of interactive real-time segmentation.

1. Introduction

There has been a recent interest in using Random Forests (RFs) [6] for computer vision problems. RFs have demonstrated to be better or at least comparable to other state-of-the-art methods in both classification [6, 4], semantic segmentation [21], real-time keypoint recognition [13] and clustering applications [14]. While yielding state-of-the-art results, RFs possess several properties that make them particularly interesting for computer vision applications. First, they are very fast in both training and classification. Second, they can be easy parallelized, which makes them interesting for multi-core and GPU implementations [20]. Additionally, RFs are inherently multi-class, therefore they do not require to build several binary classifiers for a multi-class problem. Compared to boosting and other ensemble methods, RFs are also more robust against label noise [6].

Usually random forests are trained in off-line mode, *i.e.*, the entire training data is given in advance and the training and testing phases are separated. However in practice, training data may not be given in advance but arrives sequentially, for instance in tracking applications where predictions are required on-the-fly. In such situations, learning algorithms have to be able to work in an on-line mode[1]. On-line learning has numerous advantages over off-line methods: *e.g.*, memory requirements are much lower because samples do not need to be stored, a huge amount of available data can be exploited by on-line methods, the training is usually much faster, off-line methods are not applicable if the data generation process is on-line or the underlying distribution changes over time.

RFs are ensembles of randomized decision trees combined using bagging. Hence, for an on-line version one has to combine on-line bagging [15] and on-line decision trees with random feature-selection. There exist incremental methods for single decision trees but they are either memory intensive, because every node sees and stores all the data [22], or have to discard important information if parent nodes change. The recursive nature of decision trees makes on-line learning a difficult task because due to the hard splitting rule, errors cannot be corrected further down the tree. Some methods alleviate this problem by combining decision trees with ideas from neural networks [2] but have the disadvantage that they usually lose the $\mathcal{O}(\log n)$ evaluation time because samples are propagated to all nodes[2].

[1] In this work, we distinguish "on-line" from "incremental" learning. On-line has to discard a sample after learning (no memory) and unlike to incremental learning is not allowed to store it.

[2] Additionally, we would like to mention that there exists a method proposed twice by Osman *et al.* called "On-line Incremental Random Forests", ICMV, 2007 as well as Elgawi *et al.* called "Online Random forests based on corrfs and corrbe", OLCV, 2008. We do not cite this method in the usual way because in contrast as suggested in the title it is not really an on-

2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops
978-1-4244-4441-0/09/$25.00 ©2009 IEEE

In this paper, we propose a novel on-line algorithm for random forests that has neither of the previous problems. We combine ideas from on-line bagging and extremely randomized forests. This allows us to circumvent the need for recursive discarding of estimated statistics while still allowing for sample evaluation in $\mathcal{O}(\log n)$. Additionally, our algorithm allows for temporal weighting of knowledge by discarding entire trees (based on their estimated out-of-bag errors) in fixed time intervals and consecutive growing of new trees. This increases the adaptivity and can be useful, for instance, in applications with temporal noise or high appearance changes during visual tracking.

Additionally, our algorithm can solve multi-class problems without a need for common binary decompositions, *e.g.*, 1-vs-all. As it has been discussed by Saffari *et al.* [18] such binary decompositions have some drawbacks: 1) The computational burden is higher since we need to build several binary classifiers, 2) Such decompositions usually leads to very unbalanced datasets where the majority of the classes are from the negative class, 3) Since each binary classifier is built independently, their real valued outputs might not be directly comparable. However, our algorithm is able to build on-line multi-class classifiers and therefore is not prone to such problems.

We show on standard machine learning problems that our algorithm performs comparable to common off-line RFs. Additionally, we demonstrate the suitability of our method on two well-suited computer vision tasks: First, we apply ORFs to visual tracking using simple features. Compared to other adaptive tracking-by-detection methods, *e.g.*, on-line boosting [11], we show that our algorithm is more stable and more resistant to occlusions due to higher noise tolerance and the possibility to forget information by controlled discarding of entire trees. Second, we use ORFs for real-time interactive segmentation, based on our recent work presented in [19] and show that ORFs deliver identical results as common batch RFs, however, without the need of storing any samples.

In the following Section 2, we shortly review random forests and then in detail present our on-line algorithm. Section 3 delivers several experiments on both machine learning and tracking tasks. Finally, the paper concludes with Section 4.

2. On-line Random Forests

Each tree in a forest is built and tested independently from other trees, hence the overall training and testing procedures can be performed in parallel. During the training, each tree receives a new bootstrapped training set generated by sub-sampling with replacement of the original training set. We refer to those samples which are not included during the training of a tree as the *Out-Of-Bag* (OOB) samples of that tree. These samples can be used to compute the *Out-Of-Bag-Error* (OOBE) of the tree as well as the ensemble which is an low-biased estimate of the generalization error [5].

The tests at each decision node of the tree are selected by first creating a set of random tests and then picking the best among them according to some quality measurement (*e.g.*, information gain or Gini index). The trees are usually grown to their full size without pruning.

We denote the t^{th} tree of the ensemble as $f(x, \theta_t) : \mathcal{X} \rightarrow \mathcal{Y}$, where θ_t is a random vector capturing the various stochastic elements of the tree (such as the randomly subsampled training set or selected random tests at its decision nodes). For notation brevity, we usually represent a tree as $f_t(x) = f(x, \theta_t)$. We also denote the entire forest as $\mathcal{F} = \{f_1, \cdots, f_T\}$, where T is the number of trees in the forest. We can write the estimated probability for predicting class k for a sample as

$$p(k|x) = \frac{1}{T} \sum_{t=1}^{T} p_t(k|x), \tag{1}$$

where $p_t(k|x)$ is the estimated density of class labels of the leaf of the t^{th} tree where x falls. The final multi-class decision function of the forest is defined as

$$C(x) = \arg\max_{k \in \mathcal{Y}} p(k|x). \tag{2}$$

Breiman [6] defined the classification *margin* of a labeled sample (x, y) as

$$m_l(x, y) = p(y|x) - \max_{\substack{k \in \mathcal{Y} \\ k \neq y}} p(k|x). \tag{3}$$

It is obvious that for a correct classification $m_l(x, y) > 0$ should hold. Therefore, the generalization error is given by

$$GE = E_{(X,Y)}(m_l(x, y) < 0), \tag{4}$$

where the expectation is measured over the entire distribution of (x, y). It has also been shown by Breiman [6] that this error has an upper bound in form of

$$GE \leq \bar{\rho} \frac{1 - s^2}{s^2}, \tag{5}$$

where $\bar{\rho}$ is the mean correlation between pairs of trees in the forest [3] and s is the strength of the ensemble (*i.e.*, the expected value of the margin over the entire distribution). In other words, for a low generalization error the individual trees should be highly independent and have high accuracy.

line method and most importantly, there exist reasonable suspicions that the key contributions of the work are taken from other authors *e.g.*, from [2] without citation.

[3] The correlation is measured on how much similar their predictions are.

2.1. On-line Learning

The original RF algorithm as described above is designed to learn in batch or off-line mode, *i.e.*, each tree is trained on a full sub-set of labeled samples drawn from $\mathcal{X}$. To make the algorithm operate in on-line mode, there are two main questions to be answered: 1) How to perform bagging in on-line mode? 2) How to grow random trees on-the-fly? We discuss the proposed solutions to these questions in next sections.

2.1.1 On-line Bagging

For the bagging part, we use the method proposed by Oza *et al.* [15] where the sequential arrival of the data is modeled by a Poisson distribution. Each tree $f_t(x)$ is updated on each sample k times in a row where k is a random number generated by Poisson(λ) and λ is usually set to a constant number, in our case equal to one. Oza proved convergence of this method to off-line bagging.

2.1.2 On-line Random Decision Trees

Each decision node in a tree contains a test in form of $g(x) > \theta$. These tests usually contain two main parts: 1) a randomly generated test function, $g(x)$ which usually returns a scalar value, 2) a threshold θ which based on the random feature decides the left/right propagation of samples. In off-line mode, RFs select randomly a set of such tests and then pick the best according to a quality measurement. If the threshold is also chosen randomly, the resulting RF is usually refered to *Extremely Randomized Forest* [10].

In on-line mode, we grow extremely randomized trees by generating the test functions and thresholds randomly. During growing of a randomized tree, each decision node randomly creates a set of tests and picks the best according to a quality measurement. Usual choices for quality measures are the entropy ($L(\mathcal{R}_j) = -\sum_{i=1}^{K} p_i^j \log(p_i^j)$) or the Gini index ($L(\mathcal{R}_j) = \sum_{i=1}^{K} p_i^j(1-p_i^j)$), where p_i^j is the label density of class i in node j and K is the number of classes. Computing such quality measures depends mainly on the estimation of the label densities, which can be performed in on-line mode.

More specifically, when a node is created it creates a set of N random tests $\mathcal{S} = \{(g_1(x), \theta_1), \ldots, (g_N(x), \theta_N)\}$. This node then starts to collect the statistics of the samples falling in it. It also maintains the statisitcs of the splits made with each test in $\mathcal{S}$. Denote by $\mathbf{p}_j = [p_1^j, \ldots, p_K^j]$ the statistics of class labels in node j. For a random test $s \in \mathcal{S}$, two sets of statistics are also collected: $\mathbf{p}_{jls} = [p_1^{jls}, \ldots, p_K^{jls}]$ and $\mathbf{p}_{jrs} = [p_1^{jrs}, \ldots, p_K^{jrs}]$ corresponding to the statistics of samples falling into left (l) and right (r) partitions according to test s.

The gain with respect to a test s can be measured as:

$$\Delta L(\mathcal{R}_j, s) = L(\mathcal{R}_j) - \frac{|\mathcal{R}_{jls}|}{|\mathcal{R}_j|} L(\mathcal{R}_{jls}) - \frac{|\mathcal{R}_{jrs}|}{|\mathcal{R}_j|} L(\mathcal{R}_{jrs}), \tag{6}$$

where $\mathcal{R}_{jls}$ and $\mathcal{R}_{jrs}$ are the left and right partitions made by the test s and $|.|$ denotes the number of samples in a partition. Note that $\Delta L(\mathcal{R}_j, s) \geq 0$. A test with higher gain, produces better splits of the data with respect reducing the impurity of a node. Therefore, when splitting a node, the test with highest gain is chosen as the main decision test of that node.

When operating in the off-line mode, the decision node has access to all the data falling to that node, and therefore has a more robust estimate of these statistics, compared to a node operating in on-line mode. In the on-line mode, the statistics are gathered over time, therefore, the decision when to split depends on 1) if there has been enough samples in a node to have a robust statistics and, 2) if the splits are good enough for the classification purpose. Because, the statistics of the subsequent children nodes are based on this selection and since the errors in this stage cannot be corrected further down the tree when we already made a decision, we need to develop a method which can tell the node when it is appropriate to perform a split.

Therefore, we propose the following non-recursive strategy for on-line learning of the random decision trees: A newly generated tree starts with only one root node with a set of randomly selected tests. For each test in the node we gather the statistics on-line. We introduce two hyper-parameters: 1) the minimum number of samples a node has to see before splitting α, 2) the minimum gain a split has to achieve β. Thus, a node splits when $|\mathcal{R}_j| > \alpha$ and $\exists s \in \mathcal{S} : \Delta L(\mathcal{R}_j, s) > \beta$.

After a split occured, we propagate the $\mathbf{p}_{jls}$ and $\mathbf{p}_{jrs}$ to the subsequent newly generated left and right leaf nodes, respectively. This way, a new node starts already with the knowledge of its parent nodes, and therefore, can also perform classification on-the-fly even without observing a new sample. The entire on-line RF algorithm is depicted in Algorithm 1.

Note that this tree-growing strategy is similar to that of evolving trees (ETrees) [16]. An ETree is a tree-structed self-organizing map (SOM) used in many data analysis problems. In particular, in ETrees each node counts the number of observations seen so far and splits the node after a constant threshold has been exceeded. Another similar approach to ours is that of a Hoeffding tree [9]. A Hoeffding tree is also a growing decision tree, where the split decision is made on the Hoeffding bound which theoretically guarantees that with probability $1-\rho$ the true statistical average of a random variable r is $\hat{r} - \epsilon$ with $\epsilon = \sqrt{\frac{ln(1/\rho)}{2n}}$, where n is the number of observations performed and $\hat{r}$ is the current

Algorithm 1 On-line Random Forests

Require: Sequential training example $\langle x, y \rangle$
Require: The size of the forest: T
Require: The minimum number of samples: α
Require: The minimum gain: β

// For all trees
for t from 1 to T **do**
  k $\leftarrow$ Poisson(λ)
  if $k > 0$ **then**
    // Update k times
    for u from 1 to k **do**
      $j =$ findLeaf(x).
      updateNode($j, \langle x, y \rangle$).
      if $|\mathcal{R}_j| > \alpha$ and $\exists s \in \mathcal{S} : \Delta L(\mathcal{R}_j, s) > \beta$ **then**
        Find the best test:
        $s_j = \arg\max_{s \in \mathcal{S}} \Delta L(\mathcal{R}_j, s)$.
        createLeftChild($\mathbf{p}_{jls}$)
        createRightChild($\mathbf{p}_{jrs}$)
      end if
    end for
  else
    Estimate $OOBE_t \leftarrow$ updateOOBE($\langle x, y \rangle$)
  end if
end for
Output the forest $\mathcal{F}$.

estimate of the random variable.

Although both the ETree and the Hoeffding tree would definately also be a useful choice for our splitting criterion, we believe that our approach, *i.e.*, continuously measuring the gain of a potential split, fits better to the inherent nature of decision trees.

2.1.3 Temporal Knowledge Weighting

For some applications, such as tracking, the distribution of samples might change over time. Therefore, it is required to have temporal knowledge weighting that allows unlearning old information. If the algorithm is operating in such a scenario, we allow our forest to discard the entire tree. Note that the Poisson process of on-line bagging leaves out some trees from being trained on a sample. Therefore, we can estimate the $OOBE_t$ of each tree on-line. Based on this estimate, we propose to discard trees randomly from the ensemble where the probability of discarding a tree depends on its out-of-bag-error and also its age a_t (the number of samples it has seen so far). Since in an ensemble of trees the impact of a single tree is relatively low, discarding one tree usually does not harm the performance of the entire forest. However, doing this continuously ensures adaptivity throughout time. This process is shown in Algorithm 2, where γ determines the temporal knowledge weighting rate. During the training of the forest, one tree is randomly chosen based on its age and if its $OOBE$ is large, then the tree is replaced with a new tree. In out tracking experiments, we fixed $\gamma = 0.05$.

Algorithm 2 Temporal Knowledge Weighting

Require: The knowledge weighting rate: γ

Select a tree randomly from $\{f_t | f_t \in \mathcal{F}, a_t > 1/\gamma\}$.
if $OOBE_t >$ rand() **then**
  // Discard the tree.
  $f_t =$ newTree()
end if

3. Experiments

The purpose of the experiments is to compare the performance of the novel on-line algorithm with its off-line counterpart on standard machine learning data sets and demonstrate its suitability on the task of visual object tracking [4].

3.1. Machine Learning

We use the *DNA*, *g50c*, *Letter*, *Mushrooms*, *SatImage*, and *USPS* datasets from the Semi-Supervised Benchmarks [8] and LibSVM repository [7]. A summary of these sets is presented in Table 1. For these experiments, we set the number of trees in the forest to be 100, and we select 10 random features and thresholds as decision tests. We set the $\alpha = 0.1 * N_{train}$ and $\beta = 0.1$ for the on-line random forest. Additionally, like other on-line learning methods, we continue the training process of the on-line random forest when we reach the end of the dataset by shuffling the data and repeating the whole process for 10 times. Also note that for all datasets, we use the multi-class RF. For sanity check we also report results obtained with on-line boosting for feature selection [11]. For on-line boosting, we used 50 selectors with 10 decision stumps in each selector. Each decision stump gathers the mean of the positive and negative samples and puts the classification threshold in mid-point of these two means. Also for on-line boosting we perform a 1-vs-all classification for multi-class problems.

We repeat these experiments 5 times and report the average and standard deviation of the classification error in Table 1. As it can be seen from this table, the on-line method achieves results that are very close to the off-line random forest. Figure 1 compares the classification error of the off-line and on-line trained RF when the number of training samples are changing on the *USPS* dataset. We can see that the with increasing the number of samples, the on-line RF slowly converges to the performance of the off-line RF. This

[4] Source code is available under www.ymer.org/amir/software/online-random-forests

Dataset	# Train	# Test	# Class	# Feat.	Off-line RF	On-line RF	OAB
DNA	1400	1186	3	180	0.109 ± 0.006	0.112 ± 0.008	0.173 ± 0.01
Letter	15000	5000	26	16	0.097 ± 0.014	0.104 ± 0.008	0.263 ± 0.023
Mushrooms	6000	2124	2	112	0.010 ± 0.008	0.012 ± 0.013	0.013 ± 0.001
SatImage	3104	2000	6	36	0.113 ± 0.005	0.118 ± 0.004	0.257 ± 0.11
USPS	7291	2007	10	256	0.078 ± 0.001	0.086 ± 0.005	0.224 ± 0.015

Table 1. Data sets used for the machine learning experiments, and the average classification error on the test set.

effect shows the success of the on-line training. Additionally, on-line random forests consistently outperform on-line boosting on all datasets. The poor performance of the on-line boosting can be attributed to the facts that on-line boosting is only able to operate on the binary classification problems, and therefore, is not able to capture the whole distribution of the multi-class feature space while training in on-line mode.

3.2. Tracking

In this experiment, we evaluated our on-line random forests on various publicly available tracking scenarios and compared it to a state-of-the-art tracker based on on-line boosting [11]. Since the main purpose of this experiment is the comparison of the two on-line algorithms for the tracking task, we only use simple Haar-features, did not implement any rotation and scale search and avoid any other engineering methods, although these things should definitely help improving the tracking results. Please note that [11] partly present better results than reported here which comes from the fact that they also used HOG-features and local binary patterns.

For all experiments we used 100 trees, a maximum tree-depth of 5 and used nodes with 10 random features, $\alpha = 100$, and $\beta = 0.1$. Note, however, that changing these numbers does not change the final results significantly. For the on-line boosting, we used 50 selectors with each 150 features. Both on-line boosting and our method run in real-time. However, since on-line random forests are inherently parallel, we also ported the algorithm onto a common NVidia GPU which allowed for an additional 10-times speed up leaving future space for additional algorithmus that might help improving the tracking accuracy.

3.2.1 Datasets

Our datasets consist of four publicly available sequences presenting various types of lighting, pose, scale and appearance changes. The first "Occluded Face" was taken from [1] [5]. Then, we took the famous "David Indoor" and "Sylvester" sequences from Ross *et al.* [17] and "Rotating Girl" from [3]. All sequences are grey-scale and resized to 320 x 240 pixels.

For the public datasets, we give detailed analyses and comparisons in Figure 2 and depict some representative results in in Figure 3. For further detailed tracking results we refer the reader to the supplementary material.

As can be seen, our method leads to more stable tracking results, while being highly adaptive to appearance changes. Especially when it comes to occlusion or object disappearance our method has advantages to on-line boosting due its increased noise tolerance.

3.2.2 Discussion

The complicated task in tracking using an on-line detector is to continuously self-train an appearance model while avoiding wrong updates that may cause drifting. The experiments suggest that our on-line RF algorithm is a promising choice for such a task due to the following reasons: First, in supervised learning tasks RFs have shown to achieve similar results as other popular learning algorithms such as boosting or SVMs. Our algorithm converges to the off-line version and, thus, is able to learn state-of-the-art appearance models. Second, RFs have shown to be more robust to noise, *i.e.*, wrong updates, which favours their usage in self-learning tasks where noisy samples occur inherently. Our algorithm also allows for unlearning knowledge which further makes it more robust to outliers. Third, tracking typically suffers from the stability-plasticity-dilemma [12] which means a stable model should be learned while still being highly adaptive. If the model is too inertial it cannot cope with appearance changes. If the model is too flexible it increases the potential risk of drifting. Although our algorithm suffers from the same dilemma, it seems to provide an inherent compromise that other algorithms do not have. To put it more precisely, the freezing of body nodes increases the model stability and fixes knowledge while on-line updating of leaf nodes is sufficient to achieve state-of-art on-line adaptivity. This is also confirmed by former off-line studies of Breiman who reports that the inner structure of the forest (on the decision node level) matters far less than the final decisions done by the leaf nodes.

[5] Please note that we do not compare to [1], because it uses much better features and our focus lies on the learning algorithm

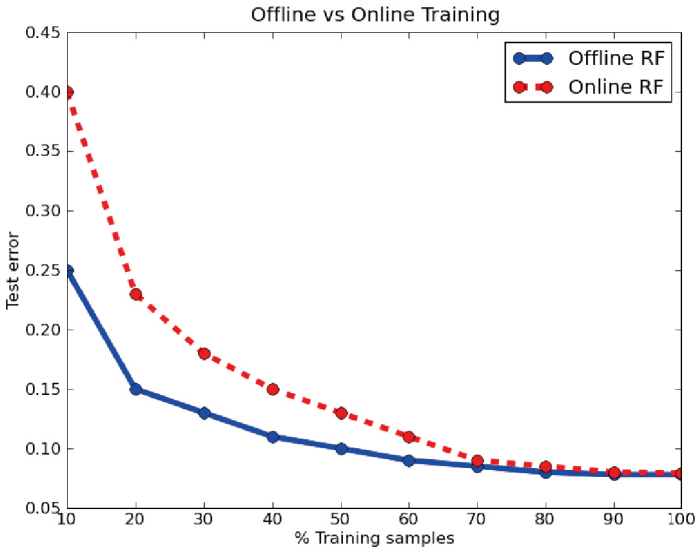
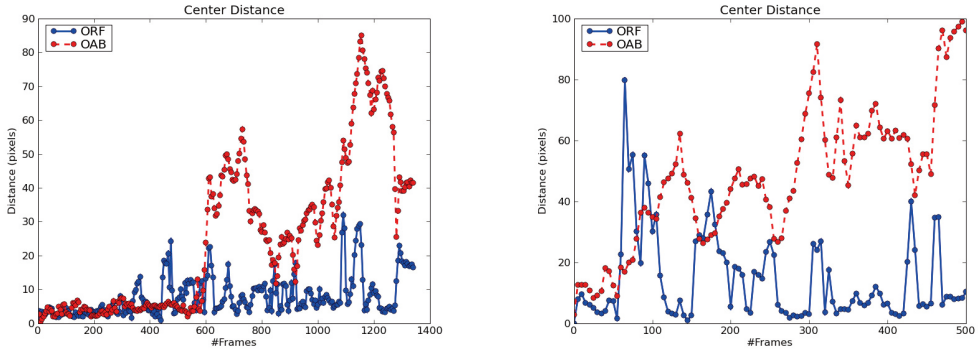
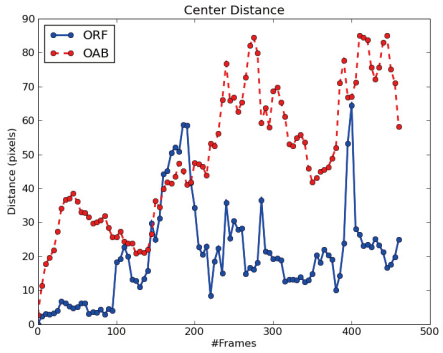


Figure 1. Classification error with respect to the ratio of labeled samples for off-line (blue) and on-line training (red dashed) with increasing number of training samples on USPS dataset. As can be seen, with increasing number of training samples the on-line learner converges to the off-line performance.

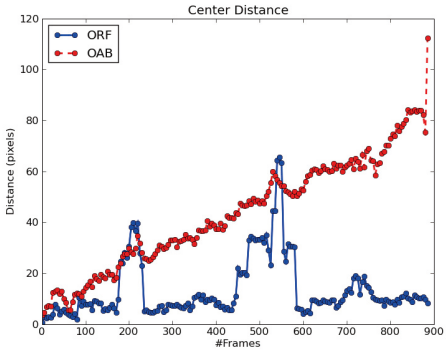


(a) Sylvester

Girl (b)



(c) David



(d) Occluded face

Figure 2. Comparison of On-line Random Forests (ORF) with On-line AdaBoost (OAB) on four state-of-the-art tracking sequences. As can be seen, our method significantly outperforms boosting on all sequences.

Figure 3. Comparison of ORFs and OAB on four public sequences.

3.3. Interactive Segmentation

In this experiment, we applied our algorithm to the task of interactive segmentation. In interactive segmentation, one wants to semi-automatically separate a foreground region from the background with the help of user input. Recently, Santner *et al.* [19] showed that this task can be performed effectively using Random Forest to train a discriminative prior model. This model is then plugged into a weighted Total Variation based segmentation algorithm. Both the RF and the segmentation are implemented on a GPU. However, after each user input usually the model has to be retrained from scratch. Hence, exchanging the off-line RF with an on-line learner can further speed-up the process in order to increase user convenience. In the following, we give some representative results for interactive segmentation using the approach proposed in [19] and our on-line random forests with 100 trees.

As can be seen in Figure 4, our method is able to deliver high quality segmentation results. Please note that the results in fact are identical to the off-line model and are thus skipped here. For further details about the approach we refer the reader to [19].

4. Conclusion

This paper introduced a novel on-line random forest algorithm. Therefore, we combined on-line bagging, random feature selection and a novel tree-growing method that allows for on-line building of decision trees. Experiments on machine learning data show that the method is converging to its off-line counterpart. Additionally, we demonstrated the usability of our method on the task of visual tracking and interactive image segmentation. Our method is quite stable, runs in real-time and is easy to implement.

In future work we plan to provide a formal proof of convergence for our method. Furthermore, we want to apply the algorithm to additional computer vision applications.

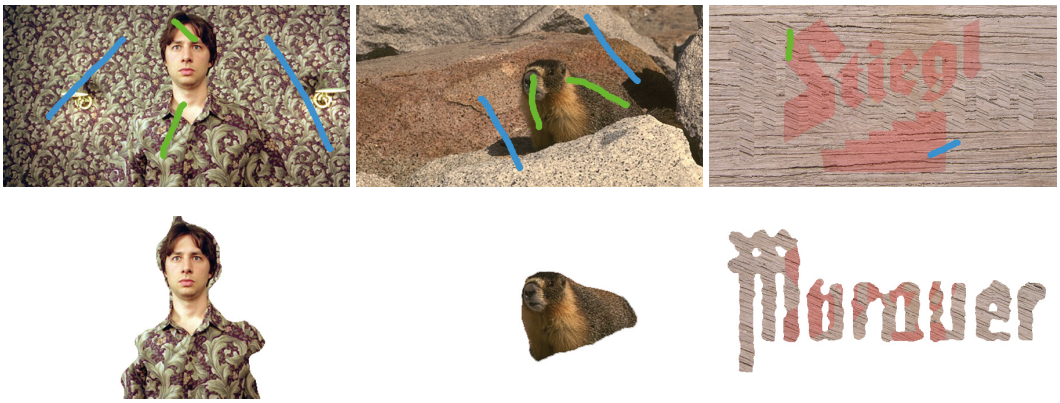


Figure 4. Interactive segmentation results using on-line random forests and a Total Variation based segmentation algorithm. Note that the green marks denote the foreground and blue marks show the background annotations by the user.

Acknowledgment

This work was supported by the Austrian Joint Research Project Cognitive Vision under projects S9103-N04 and S9104-N04 and the Austrian Science Fund (FWF P18600), by the FFG projects AUTOVISTA (813395) and EVis (813399) under the FIT-IT programme.

References

[1] A. Adam, E. Rivlin, and I. Shimshoni. Robust fragments-based tracking using the integral histogram. In *Proc. CVPR*, 2006.

[2] J. Basak. Online adaptive decision trees: Pattern classification and function approximation. *Neural Comput.*, 18:2062–2101, 2004.

[3] S. Birchfield. Elliptical head tracking using intensity gradients and color histograms. In *Proc. CVPR*, 1998.

[4] A. Bosch, A. Zisserman, and X. Munoz. Representing shape with a spatial pyramid kernel. In *Proceedings of the ACM International Conference on Image and Video Retrieval (CIVR)*, pages 401–408, 2007.

[5] L. Breiman. Out-of-bag estimates. Technical report, 1996.

[6] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, October 2001.

[7] C. C. Chang and C. J. Lin. Libsvm: a library for support vector machines, 2001.

[8] O. Chapelle, B. Schölkopf, and A. Zien. *Semi-Supervised Learning*. Cambridge, MA, 2006.

[9] P. Domingos and G. Hulten. Mining high-speed data streams. In *Knowledge Discovery and Data Mining*, 2000.

[10] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. In *Machine Learning*, volume 63, pages 3–42, 2006.

[11] H. Grabner and H. Bischof. On-line boosting and vision. In *Proc. CVPR*, volume 1, pages 260–267, 2006.

[12] S. Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Neural networks and natural intelligence*, pages 213–250, 1998.

[13] V. Lepetit, P. Lagger, and P. Fua. Randomized trees for real-time keypoint recognition. In *Proc. CVPR*, volume 2, pages 775–781, 2005.

[14] F. Moosmann, B. Triggs, and F. Jurie. Fast discriminative visual codebooks using randomized clustering forests. In *Advances in Neural Information Processing Systems 19*, pages 985–992, 2006.

[15] N. Oza and S. Russell. Online bagging and boosting. In *Proceedings Artificial Intelligence and Statistics*, pages 105–112, 2001.

[16] J. Pakkanen, J. Iivarinen, and E. Oja. The evolving tree—a novel self-organizing network for data analysis. *Neural Process. Lett.*, 2004.

[17] D. Ross, J. Lim, and M. Yang. Adaptive proballistic visual tracking with incremental subspace update. In *Proc. ECCV*, volume 2, pages 470–482, 2004.

[18] A. Saffari, C. Leistner, and H. Bischof. Regularized multi-class semi-supervised boosting. In *IEEE Conference on Computer Vision and Pattern Recogntion*, 2009.

[19] J. Santner, M. Unger, T. Pock, C. Leistner, A. Saffari, and H. Bischof. Interactive texture segmentation using random forests and total variation. In *Proceedings of the British Machine Vision Conference (BMVC)*, London, UK, September 2009. to appear.

[20] T. Sharp. Implementing decision trees and forests on a gpu. In *ECCV*, pages 595–608, 2008.

[21] J. Shotton, M. Johnson, and R. Cipolla. Semantic texton forests for image categorization and segmentation. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, 2008.

[22] E. Utgoff, N. Bergman, and J. Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 1997.