An Introduction to Random Forests for Multi-class Object Detection

Juergen Gall, Nima Razavi, and Luc Van Gool

[1] Computer Vision Laboratory, ETH Zurich,
{gall,nrazavi,vangool}@vision.ee.ethz.ch
[2] ESAT/IBBT, Katholieke Universiteit Leuven

Abstract. Object detection in large-scale real-world scenes requires efficient multi-class detection approaches. Random forests have been shown to handle large training datasets and many classes for object detection efficiently. The most prominent example is the commercial application of random forests for gaming [36]. In this chapter, we describe the general framework of random forests for multi-class object detection in images and give an overview of recent developments and implementation details that are relevant for practitioners.

Keywords: multi-class object detection, Hough forest, regression forest, random forest

1 Introduction

Object detection for real world applications is still a challenging problem. While recent research datasets like PASCAL VOC [11], ImageNet [9], or the Caltech Pedestrian Dataset [10] increase the amount of training and testing examples to get closer to real world problems, the ability of detectors to process large data sets in reasonable time becomes another important issue besides accuracy. It is not only the number of training examples that matters, but also the number of classes.

A family of methods that can handle large amount of training data efficiently and that are inherently suited for multi-class problems are based on random forests [1, 5]. Random forests are ensembles of randomized decision trees that can be applied for regression [8, 12, 18], classification tasks [25, 27, 29, 6, 39, 4, 34, 37, 36], and even both at the same time [15, 30, 38, 17, 13]. The most prominent application of random forest is the detection of human body parts from depth data [36]. The method was trained on 900k training examples to detect 31 body parts (classes) and runs at around 200 frames per second on the Xbox GPU. This commercial application demonstrates the practicability of random forests for large-scale real-world computer vision problems.

The scope of this chapter is to give an introduction to random forests in the context of multi-class object detection and to give an overview of recent developments. For a more general discussion on random forests, we refer to the book [5] and the tutorial [7]. Rather than providing a detailed experimental

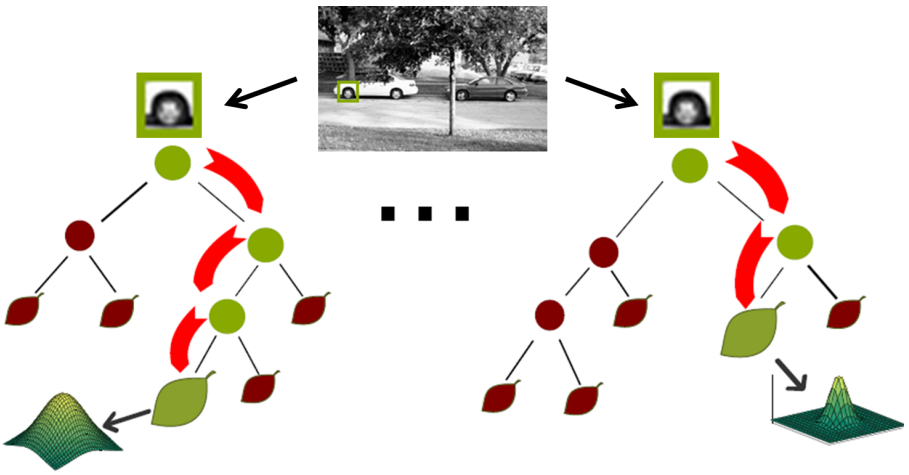


Fig. 1. A random forest consists of a set of trees that map an image patch to a distribution stored at each leaf. The spheres indicate split nodes that evaluate the appearance of a patch and pass it to the right or left child until a leaf is reached.

evaluation which has been already presented in the referenced works, the chapter serves more as a guide for practitioners.

2 Random Forests for Object Detection

A random forest consist of a set of trees T_t where each tree consists of split nodes and leaves as illustrated in Figure 1. The split nodes evaluate each arriving image patch and, depending on the appearance of the patch, pass it to the left or right child. Each leaf L stores the statistics of the image patches that arrived during training. For a classification task, it is the probability for each class c, denoted by $p(c|L)$. For a regression task, it is a distribution over the continuous parameter $x \in \mathbb{R}^H$ that one wants to estimate. While image segmentation is a typical classification task where one wants to estimate the class label for each image patch, object localization can be regarded as a regression problem where each patch of the object predicts the location of the object in the image. Since object detection involves both classifying patches belonging to an object and using them to regress the location and scale of the object, random forests for object detection need to be trained to satisfy both objectives.

2.1 Training

For training, a set of images is collected where each object is annotated by a bounding box and the class label c. The background images are only annotated by the class label. In order to handle large amount of training data and to avoid overfitting, randomness is introduced by training each tree on a randomly

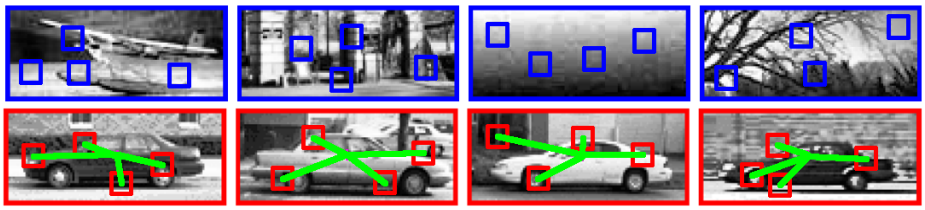


Fig. 2. For training, a subset of image patches is taken from the entire training set. In the simplest case, there are only two classes; one containing negative or background examples (*blue*) and another containing positive examples (*red*). While the class labels are required to distinguish object patches from background patches (*classification*), additional offset vectors of the positive patches to the center of the object are stored (*green*). The offset vectors will be used to predict the location of the object (*regression*).

sampled subset of the training data [5]. For object detection, this means to select a random set of images for each class and to sample a subset of patches for each image as illustrated in Figure 2. For each sampled patch $\mathcal{P}_i$ that does not belong to the background, the offset to a reference point of the object $\boldsymbol{d}_i$ is stored. Ideally, the reference point is always the same for all training instances of a class, e.g., the head of a pedestrian. However, taking the center of the bounding box as reference point is usually a more practical choice. In general, the reference point does not need to be the center of the object, but it should be as consistent as possible among training examples. Scale is handled during testing and the positive examples are scaled to a unit size s_u. A good choice for object detection has been to use image patches of size 16×16 pixels and scale the images such that the longest spatial dimension of the bounding box is about 100 pixels. In this setting, a patch covers meaningful parts like a wheel of a car or the head of a human as shown in Figures 2 and 4. In case of tight bounding boxes around the objects, it is beneficial to consider all patches for sampling that have the patch center inside of a bounding box. In this way, important boundary information can be better captured [**?**].

In summary, we have a set of training patches $\{\mathcal{P}_i = (\mathcal{I}_i, c_i, \boldsymbol{d}_i)\}$ that are randomly sampled from the examples where,

$\mathcal{I}_i$ are the extracted image features of the patch,
c_i is the class label for the exemplar, the patch is sampled from,
$\boldsymbol{d}_i$ is a offset vector from the patch center to the reference point.

Patches sampled from background images have a pseudo offset, i.e., $\boldsymbol{d}_i = 0$. We denote the set of randomly sampled training patches for a tree T_t by $A = \{\mathcal{P}_i\}$.

In order to train a tree that can be used for object detection, one has to find a split function

$$f_\phi(\mathcal{P}) \in \{0, 1\} \tag{1}$$

for each non-leaf node that separates the training patches in an optimal way. The split functions are therefore also termed as weak learners [7]. The split function

evaluates one or more image features of the patch $\mathcal{P}$ and sends it to the left ($f_\phi(\mathcal{P}) = 0$) or right child ($f_\phi(\mathcal{P}) = 1$) of the node; see Figure 1. The split functions are parametrized by a set of parameters ϕ that need to be optimized during training.

Each tree can be trained in parallel using the general random forest framework [5]. Starting at the root node with the training set $A_{node} = A$, a tree grows recursively:

1. Generate a random set of parameters $\Phi = \{\phi_k\}$.
2. Divide the set of patches A_{node} into two subset A_L and A_R for each $\phi \in \Phi$:
$$A_L(\phi) = \{\mathcal{P} \in A_{node} | f_\phi(P) = 0\} \tag{2}$$
$$A_R(\phi) = \{\mathcal{P} \in A_{node} | f_\phi(P) = 1\} \tag{3}$$
3. Select the splitting parameters ϕ^* that maximize the gain g:
$$\phi^* = \underset{\phi \in \Phi}{\operatorname{argmax}}\, g(\phi, A_{node}) \tag{4}$$
where
$$g(\phi, A_{node}) = \mathcal{H}(A_{node}) - \sum_{S \in \{L,R\}} \frac{|A_S(\phi)|}{|A_{node}|} \mathcal{H}(A_S(\phi)). \tag{5}$$
Depending on the task, $\mathcal{H}(A)$ is chosen such that g measures the gain of the classification or regression performance of the children in comparison to the current node.
4. Continue growing with the training subsets A_L and A_R if the stopping criteria are not satisfied; otherwise, create a leaf node and store the statistics of the training data A_{node}.

Step 1 is another source of randomness that reduces training time whereas evaluating all parameters ϕ would be infeasible in many cases. While the family of splitting functions f_ϕ, the measure $\mathcal{H}$, and the stopping criteria will be discussed in Section 3, we continue with the prediction model stored at each leaf L.

In the context of object detection, we are interested in the class probability and the spatial distribution of the training patches for each class. The class probability $p(c|L)$ can be estimated by
$$p(c|L) = \frac{|A_c^L| \cdot r_c}{\sum_c (|A_c^L| \cdot r_c)}; \quad r_c = \frac{|A|}{|A_c|} \tag{6}$$
where A^L is the set of training patches reaching the leaf L after training, A the entire training set used for training the tree, and A_c the patches in A with class label c. The factor r_c compensates for the sample bias that might have been introduced when the number of training examples is not well distributed among classes. The spatial distribution for each class, $p(\boldsymbol{d}|c, L)$, is obtained by estimating the continuous distribution from the offset samples $\boldsymbol{d} \in D_c^L$ of the patches A_c^L. While more details will be given in Section 3, the statistics of a few example leaves are shown in Figure 3.

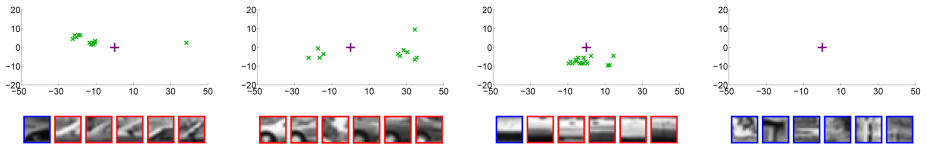


Fig. 3. Visualization of some leaves of a tree for detecting cars (side-view; two classes). Each leaf node L stores the probability of a patch belonging to the object class $p(c|L)$, estimated by the proportion of patches from the positive (*red*) and negative examples (*blue*) reaching the leaf during training. For the positive class, the offset vectors $\boldsymbol{d} \in D_c^L$ are shown (*green*). The underlying distribution $p(\boldsymbol{d}|c, L)$ is multimodal. The positive training examples falling inside each of the first three leaves can be associated with different parts of a car. The last leaf contains only negative patches. The image has been taken from [17].

2.2 Detection

For detecting an object, image patches are sampled from a test image and passed through the trees as shown in Figure 1. The image patches can be densely sampled or subsampled as for training. Each patch $\mathcal{P}(\boldsymbol{y})$ sampled from image location $\boldsymbol{y}$ ends in a leaf $L_t(\boldsymbol{y})$ for each tree T_t. In order to locate an object in the image, we evaluate the probability of an object hypothesis $\boldsymbol{h}(c, \boldsymbol{x}, s)$, i.e., the probability of an object belonging to class c with size s and its reference point at $\boldsymbol{x}$. Besides of scale, additional parameters of the object like depth [38], viewpoint [31], or aspect ratio [15] can be estimated.

The probability $p(\boldsymbol{h}|L_t(\boldsymbol{y}))$ for a single patch and a single tree is then given by

$$p(\boldsymbol{h}(c, \boldsymbol{x}, s)|L_t(\boldsymbol{y})) = p\left(\boldsymbol{d}(\boldsymbol{x}, \boldsymbol{y}, s)|c, L_t(\boldsymbol{y})\right) p\left(c|L_t(\boldsymbol{y})\right) \tag{7}$$

where

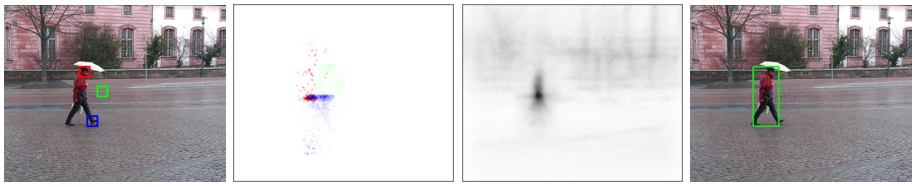
$$\boldsymbol{d}(\boldsymbol{x}, \boldsymbol{y}, s) = \frac{s_u(\boldsymbol{y} - \boldsymbol{x})}{s}. \tag{8}$$

The term $\boldsymbol{d}(\boldsymbol{x}, \boldsymbol{y}, s)$ is basically the offset between $\boldsymbol{y}$ and $\boldsymbol{x}$ given the hypothesis size s. Note that the unit size s_u and the two probabilities $p(\boldsymbol{d}|c, L_t(\boldsymbol{y}))$ and $p(c|L_t(\boldsymbol{y}))$ are known from training. The derivation of Equation (7) is straightforward and given in [17]. While random regression forests or classification forests model only one of the two terms in Equation (7), the distribution $p(\boldsymbol{h}|L_t(\boldsymbol{y}))$ combines both the regression and the classification objective.

The distribution $p(\boldsymbol{d}|c, L_t(\boldsymbol{y}))$ can be modeled by a set of votes $\boldsymbol{d} \in D_c^{L_t(\boldsymbol{y})}$. In this case, Equation (7) becomes

$$p(\boldsymbol{h}(c, \boldsymbol{x}, s)|L_t(\boldsymbol{y})) = \frac{1}{\left|D_c^{L_t(\boldsymbol{y})}\right|} \left(\sum_{\boldsymbol{d} \in D_c^{L_t(\boldsymbol{y})}} \delta_d \left(\frac{s_u(\boldsymbol{y} - \boldsymbol{x})}{s} \right) \right) p\left(c|L_t(\boldsymbol{y})\right), \tag{9}$$

where δ is a Dirac measure. Therefore, regression trees are also termed Hough forests [15] in the context of object detection. Figures 4 (a) and (b) show the votes



(a) (b) (c) (d)

Fig. 4. For each of the three patches emphasized in (a), the random forest trained on pedestrians casts weighted votes about the possible location of a pedestrian (b) (each color channel corresponds to the vote of a sample patch). Note the weakness of the vote from the background patch (green). After the votes from all patches are aggregated (c) (black corresponds to a high value), the pedestrian can be detected (d) by searching the mode of (c). The images have been taken from [17].

or distribution of three patches. While the head (red) patch yields a distribution with one strong mode, the patch of the right feet (blue) is similar to the left feet in appearance and thus yielding a distribution with two modes. The impact of the class probability can be observed for the background patch (green). Since the probability of this patch belonging to the object is close to zero, the votes are barely visible.

While Equation (7) models the probability for a single tree, the probabilities of all trees are averaged, i.e.,

$$p(\boldsymbol{h}(c, \boldsymbol{x}, s)|\mathcal{P}(\boldsymbol{y})) = \frac{1}{|\{T_t\}|} \sum_t p(\boldsymbol{h}(c, \boldsymbol{x}, s)|L_t(\boldsymbol{y})). \tag{10}$$

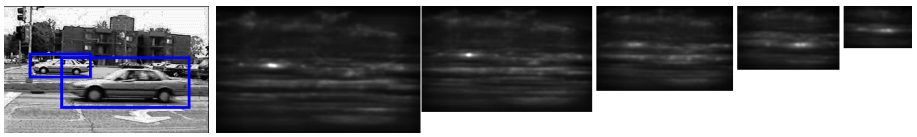
Alternatively, the probabilities can also be multiplied but averaging is more efficient [7]. Similarly, the distributions over all image patches can be either accumulated as in [17]:

$$p(\boldsymbol{h}(c, \boldsymbol{x}, s)|\mathcal{I}) = \frac{1}{|\Omega|} \sum_{\boldsymbol{y} \in \Omega} p(\boldsymbol{h}(c, \boldsymbol{x}, s)|\mathcal{P}(\boldsymbol{y})); \tag{11}$$

or multiplied as in [2]. An example using Equation (11) for a single scale is shown in Figure 4 (c). Multiple scales can be handled by processing the image at different scales as shown in Figure 5. In order to detect an object of size s, giving the training size s_u, the image is scaled by $\frac{s_u}{s}$. In this way, the scale factor in Equation (8) is already taken into account. Object detection can then performed by using mean shift to detect the modes of Equation (11); see Figures 4 and 5.

3 Implementation Details

So far, the general framework has been described. In this section, we discuss variations and implementation details that are relevant for practitioners.



(a) (b)

Fig. 5. In order to detect objects at different scales (a), the original image is scaled by the inverse expected sizes (b). The modes are detected in the joint space of image location and scale (white corresponds to a high value). The small car yields a peak on the two left images and the large car yields a peak on the right images.

3.1 Features and Binary Tests

Actually any kind of image feature can be used that is useful for object detection. This includes sparse features like SIFT [26] or SURF [3], but usually one relies on low-level features like color, gradients, or Gabor filters that can be efficiently computed. In contrast to manual designed feature descriptors, the random forest selects a split function (1) at each non-leaf node during training. All leaves of the trees can therefore be regarded as features and the split functions that are passed by a feature to reach the leaf as descriptor of the feature. However, the features and descriptors can be directly optimized for the task of object detection. A set of features obtained from simple pixel tests using intensity and first-order gradients are shown in Figure 3. The used pixel tests are defined by:

$$f_\phi(\mathcal{P}) = \begin{cases} 0 & \text{if } I^f(\boldsymbol{p}) - I^f(\boldsymbol{q}) < \tau \\ 1 & \text{otherwise.} \end{cases} \tag{12}$$

where the parameters $\phi = \{\boldsymbol{p}, \boldsymbol{q}, f, \tau\}$ comprise two pixel locations within the patch, a low-level image feature I^f of the patch, and a threshold τ. The pixel differences introduce invariance with respect to a constant change of the image features I^f. In [27], only a pixel value is thresholded, i.e., $I^f(\boldsymbol{p}) < \tau$. More general tests than (12) have been used in [8, 12], where the feature values over two regions Q and P are averaged:

$$f_\phi(\mathcal{P}) = \begin{cases} 0 & \text{if } \frac{1}{|P|}\sum_{\boldsymbol{p}\in P} I^f(\boldsymbol{p}) - \frac{1}{|Q|}\sum_{\boldsymbol{q}\in Q} I^f(\boldsymbol{q}) < \tau \\ 1 & \text{otherwise.} \end{cases} \tag{13}$$

The regions are rectangles within the patch such that the average can be efficiently computed by integral images. In general, several features can be also combined for a single test:

$$f_\phi(\mathcal{P}) = \begin{cases} 0 & \text{if } \tau_1 < \sum_f w_f \left(\frac{1}{|P_f|}\sum_{\boldsymbol{p}\in P_f} I^f(\boldsymbol{p})\right) < \tau_2 \\ 1 & \text{otherwise,} \end{cases} \tag{14}$$

where w_f is a weight between the features, e.g., $w_f \in \{-1, 0, 1\}$, and τ_1 is an additional threshold [7]. While more complex split functions allow a better

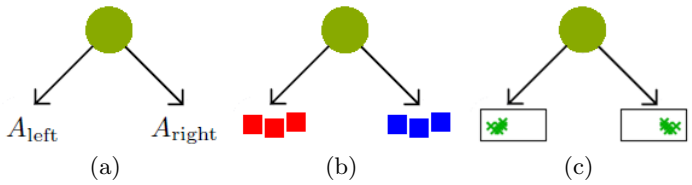


Fig. 6. (a) Each split function separates the training data at a node. (b) The classification objective aims to separate patches with different class labels. (c) The regression objective aims to maximize the localization accuracy of the offsets.

separation at each node, they also involve more parameters to estimate and increase the chance of overfitting [7]. Even for Equation (12) the patch size has an impact on the detection performance although it is not very sensitive to the exact size [17].

In practice, splitting tests of type (12) or (13) have shown to give a good performance for object detection. While in case of depth data, using only depth data already gives good results [18, 36], 32 image features have been used in [17] for object detection. Similarly to the number of parameters ϕ, the number of image features can result in overfitting, i.e., a random forest trained with less image features might perform better than a forest with many features.

To increase the robustness of features, one can introduce some variance to the patches by transforming them. For instance, patches of various sizes and orientations are used in the context of classification [27]. In [23], additional patches are tracked in video clips and added to the training data as temporal pairs. When measuring the goodness of a split, one can enforce that temporal pairs are not split. In this way, one can introduce some robustness of the features with respect to small appearance changes over time. Measuring the goodness of a split will be discussed in the next section.

In order to make the evaluation of the random set of parameters Φ more efficient, one generates the parameters of a splitting function ϕ without a threshold τ. The real-valued function $f_{\phi\setminus\tau}$ is then applied to all patches $\mathcal{P}_i \in A$, which are sorted such that $f_{\phi\setminus\tau}(\mathcal{P}_i) \leq f_{\phi\setminus\tau}(\mathcal{P}_j)$ for $i \leq j$. In this way, many thresholds can be efficiently evaluated for the splitting function f_ϕ. In [17], 10 randomly generated thresholds are generated for each of the 2000 functions $f_{\phi\setminus\tau}$, yielding 20k splitting functions for Φ in total.

3.2 Goodness of Split

Having defined a family of split functions, one has to measure the quality of a split (5) by defining $\mathcal{H}(A)$. Depending on the task, one can define a classification or regression functional; see Figure 6. An entropy-based classification functional can be computed by

$$\mathcal{H}_c(A) = -\sum_c p(c|A) \log\left(p(c|A)\right), \tag{15}$$

where $p(c|A)$ is computed as in Equation (6). The functional tends to separate patches with different class labels in order to get leaves with low uncertainty for $p(c|L)$.

As in [7], one can also define a regression functional in a similar way by:

$$\mathcal{H}_r(A) = -\sum_c \frac{1}{|A|} \sum_{\mathcal{P} \in A} \int_{\boldsymbol{d}} p(\boldsymbol{d}|c, \mathcal{P}) \log\left(p(\boldsymbol{d}|c, \mathcal{P})\right) \mathrm{d}\boldsymbol{d}, \tag{16}$$

to obtain leaves with a low uncertainty for $p(\boldsymbol{d}|c, L)$. While $\mathcal{H}_r(A)$ can be efficiently computed under the assumption that $p(\boldsymbol{d}|c, \mathcal{P})$ are Gaussian distributions, the functional becomes too expensive for more general distributions. In [17], a functional that is more efficient to compute has been used for regression:

$$\mathcal{H}_r(A) = \sum_c \left(\sum_{\boldsymbol{d} \in D_c^A} \left\| \boldsymbol{d} - \frac{1}{|D_c^A|} \sum_{\boldsymbol{d}' \in D_c^A} \boldsymbol{d}' \right\|^2 \right). \tag{17}$$

Using functional (16) with a Gaussian assumption or functional (17) is not optimal since both functionals assume a unimodal distribution of the offsets, which is not correct for object detection as illustrated in Figure 3. In practice, these approximations are, however, preferred due to training efficiency.

For object detection, one is interested in minimizing the uncertainties for $p(c|L)$ and $p(\boldsymbol{d}|c, L)$. Therefore, one searches for a splitting function f_ϕ that maximizes the gain (5) using $\mathcal{H}_c$ and $\mathcal{H}_r$, denoted by $g_c(\phi, A)$ and $g_r(\phi, A)$, respectively. While the objective is randomly selected at each node in [17], [30] uses a weighted combination of $g_c(\phi, A)$ and $g_r(\phi, A)$:

$$g_{cr}(\phi, A) = g_c(\phi, A) + w(A) g_r(\phi, A). \tag{18}$$

In [30], $w(A)$ is only defined for a two class problem with a positive and a negative class:

$$w(A) = \alpha \max\Big(p(c_{pos}|A) - t_p, 0\Big). \tag{19}$$

In general, the measure $g_{cr}(\phi, A)$ tries to separate patches with different class labels first. If the purity of positive patches exceeds a given threshold t_p, the impact of the regression functional $g_r(\phi, A)$, weighted by the constant α, increases. In [13], several weights for combining g_c and g_r based on the depth of the node and including (19) have been evaluated in the context of head pose estimation. They showed that a random approach as in [17] gives very similar performance to weighting schemes with optimized parameters. The random approach, however, does not require additional parameters and is more efficient since only one functional needs to be evaluated at each node.

While combining the classification term with the regression term improved the performance for object detection in [17], the classification term gave the best performance in the context of body part detection [18]. Body part detection is a special case since all classes are spatially connected and it seems that enforcing

a local separation based on body part labels seems to be more appropriate than making a unimodal approximation of the spatial distributions.

To avoid overfitting, the parameters of the split functions can be regularized. For instance, the weights w_f of the splitting functions (14) can be regularized by $g(\phi, A) - \lambda \sum_f \|w_f\|^2$ [28]. In [23], pairs of patches $(\mathcal{P}^1, \mathcal{P}^2)$ are used for regularization:

$$-\lambda \left(\frac{1}{|B_{pos}|} \sum_{B_{pos}} \mathbb{I}\left(f_\phi(\mathcal{P}^1) \neq f_\phi(\mathcal{P}^2)\right) + \frac{1}{|B_{neg}|} \sum_{B_{neg}} \mathbb{I}\left(f_\phi(\mathcal{P}^1) = f_\phi(\mathcal{P}^2)\right) \right), \quad (20)$$

where $\mathbb{I}$ is an indicator function. The regularizer enforces that patches that are similar under certain transformations, i.e., $(\mathcal{P}^1, \mathcal{P}^2) \in B_{pos}$, are not separated while patches that are dissimilar, B_{neg}, are separated. The regularizer can be used to introduce some robustness of the features with respect to specific transformations. In contrast to the training data A, the pairs in B do not contain class labels and can be collected from other sources. For instance, tracked patches in arbitrary video sequences were used in [23] to build pairs for regularization in the context of object detection and tracking.

Equation (20) relates to semi-supervised learning that can be implemented in an iterative approach as in [24] or by computing the unsupervised gain g_u that prefers to cluster patches of similar appearance [7]. Since the unsupervised gain does not depend on labels, it can be computed over the union of the labeled set A and an additional unlabeled set B. The supervised and unsupervised gain can be combined by:

$$g(\phi, A) + \lambda g_u(\phi, A \cup B) \quad (21)$$

where g_u is defined by using

$$\mathcal{H}_u\left(A \cup B\right) = -\int_{\mathcal{I}} p\left(\mathcal{I}|A \cup B\right) \log\left(p\left(\mathcal{I}|A \cup B\right)\right) \, \mathrm{d}\mathcal{I} \quad (22)$$

in Equation (5). As Equation (16), the term $\mathcal{H}_u$ can be efficiently computed under the assumption that the appearance of the patches of the set $A \cup B$ can be approximated by a Gaussian distribution $p(\mathcal{I}|A \cup B)$; otherwise the evaluation becomes too expensive.

Regularizers and semi-supervised learning are important when the set of labeled training data is rather small to avoid overfitting. In case of large amount of labeled training data, the set of patches does not fit in the memory and online learning [35] or subsampling strategies [35, 18] can be used. These strategies can be easily implemented using only a subset of the training data $A' \subset A$ for training a tree until a certain size. For the next step, another subset $A'' \subset A$ is sampled and passed through the previously learned tree. The training is then continued until the tree has reached a final size. After the parameters of the splitting functions at the non-leaf nodes have been optimized, the distributions at the leaves can be computed from the full training set A by passing all patches through the tree and updating the offsets D_c^L and the histograms of the class

labels $|A_c^l|$ at the leaves. On-line learning or updating the leaf statistics is also performed for object tracking [16, 35, 19] where the training examples arrive sequentially over time.

3.3 Stopping Criteria

There are three main criteria for stopping the growing of a tree. The maximum depth of a tree, a minimum number of samples arriving at a node during training $|A_{node}|$, and a threshold based on the gain measure $g(\phi^*, A_{node})$ (5). While the gain should be always strictly positive, i.e., $g(\phi^*, A_{node}) > 0$, finding a good threshold is difficult. Therefore, limiting the tree depth and the minimum number of samples are more practical criteria. In the context of on-line learning [35], it has been shown that $|A_{node}| > \epsilon$ is a sufficient criteria and an additional thresholding of the gain is not necessary. The optimal depth, however, depends on the amount of training data. For instance, the optimal performance for detecting organs in CT scans has been achieved by training 12 trees with depth 7 on the available 55 training examples [8]. In [18], 3 trees with depth 20 trained on 300k training examples performed well. While the number of trees is less critical since the performance does not decrease with more trees, trees that are too deep can have a negative impact on the performance due to overfitting [8].

3.4 Leaf Prediction Model

While $p(c|L)$ is defined in (6), there are several choices for modeling the spatial distributions $p(\boldsymbol{d}|c, L)$ at the leaf L. In [15], a Parzen estimate with a Gaussian kernel K is used to reconstruct the distribution from the samples:

$$p(\boldsymbol{d}|c, L) = \frac{1}{|D_c^L|} \left(\sum_{\boldsymbol{d}' \in D_c^L} K(\boldsymbol{d}' - \boldsymbol{d}) \right). \tag{23}$$

Although the non-parametric approach is very general, it does not scale with the number of offsets per class $|D_c^L|$. For many training examples, it is therefore recommended to approximate the distributions by a Gaussian mixture model as in [18, 20]. Since in both cases a multimodal regression functional (16) is too expansive to evaluate for training, it is therefore approximated by a more simple, unimodal measure. In case of pose estimation [8, 12], $p(\boldsymbol{d}|c, L)$ is even approximated by a single Gaussian. Although this makes the testing very efficient, it is not an appropriate choice for object detection as indicated by the leaf distributions shown in Figure 3.

3.5 Bounding Box Estimation

For getting object hypotheses, the modes of $p(\boldsymbol{h}|\mathcal{I})$ (11) can be searched by mean shift [22, 18] or by smoothing the voting space and searching for local maxima in a greedy manner [15]. In both cases, the bandwidth of the used kernel needs to

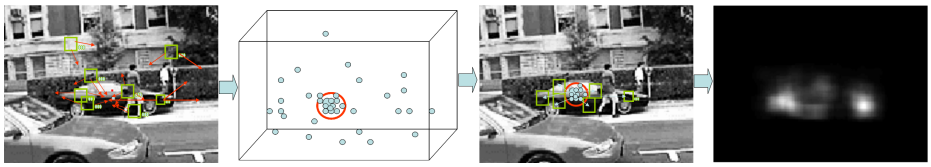


Fig. 7. Object detection with backprojection. From left to right: After passing the patches of the test image through the trees, the votes are collected. The mode of the distribution the votes are sampled from is detected by mean shift. The votes are backprojected to the image showing the image patches that voted for the object. The backprojection mask visualizes the support from the wheels of the car. Note that the occluding pedestrian is not part of the backprojection mask. The image has been taken from [31].

be large enough to detect objects where the votes do not aggregate in the exact spot. This is illustrated in Figure 7.

The computation of $p(\boldsymbol{h}|\mathcal{I})$ can be drastically reduced by sampling not all patches from the test image, but using only a subset a patches. As long as the average overlap between two nearest sampled patches is greater than 50%, the loss in performance is acceptable in comparison to the gain in runtime performance [15, 17]. In addition, one can discard leaves that are very uncertain as in [8, 12, 13, 18], i.e., if $p(c|L)$ is low or if the variance of $p(\boldsymbol{d}|c, L)$ is high. This can be achieved by using a predefined threshold or taking a fix number of the most certain leaves per image.

Having a hypothesis $\boldsymbol{h}(c, \boldsymbol{x}, s)$, the enclosing bounding box can be estimated by taking the average bounding box of the training examples of class c, after rescaling to the unit size s_u, and multiplying it by the estimated size $\frac{s}{s_u}$. The position of the bounding box is defined by $\boldsymbol{x}$.

In some cases, the aspect ratios vary widely within in a single class such that the average bounding box of the training images scaled and translated to the detection center is not precise enough. Alternatively, one can compute the backprojection of the supporting image patches for a hypothesis $\boldsymbol{h}(c, \boldsymbol{x}, s)$ [22, 31]. One approach to compute the backprojection mask extracts the maximum extent of a possible support, i.e., the largest bounding box of the training images scaled and translated to the detection center. Within the bounding box, the image patches are collected and passed again through the trees. Every time a patch votes for the hypothesis, the contribution weight of the patch $\mathcal{P}(\boldsymbol{y})$ for $\boldsymbol{h}$ is given by

$$\pi(\boldsymbol{h}(c, \boldsymbol{x}, s), \boldsymbol{y}) = \frac{1}{|\{T_t\}|} \sum_t \left(\sum_{\boldsymbol{d} \in D_c^{L_t(\boldsymbol{y})}} \frac{p\left(c|L_t(\boldsymbol{y})\right)}{\left|D_c^{L_t(\boldsymbol{y})}\right|} K\left(d - \frac{s_u(\boldsymbol{y} - \boldsymbol{x})}{s}\right) \right), \quad (24)$$

where K is the kernel used for mode detection. An obtained backprojection mask $\pi(\boldsymbol{h}, \boldsymbol{y})$ is shown in Figure 7. To obtain the bounding box, the mask can be thresholded to estimate the tightest bounding box encompassing the binary

Fig. 8. Computing the bounding box based on the backprojection. Left: annotated bounding box (blue) and estimated bounding box (green). Right: Superimposed backprojection mask and estimated bounding box.

mask. In [31], the threshold is defined by $\frac{1}{2}\max_{\boldsymbol{y}} \pi(\boldsymbol{h}, \boldsymbol{y})$. Two examples are shown in Figure 8. In [22, 33], the backprojection has been additionally augmented by segmentation masks obtained from segmented training data. The segmentation mask can also be used for verification.

In order to detect multiple instances in a single image, one can use a greedy approach. Starting with the hypothesis with the highest score $p(\boldsymbol{h}|\mathcal{I})$, the image patches that support the hypothesis are removed and the detection process continues until a maximum number of hypotheses have been extracted from the image or the remaining hypotheses have a score below a given threshold. However, there are more principled ways to detect multiple instances. In [22, 2, 33], optimization procedures for non-maximum suppression, for instance, based on the minimum description length (MDL) principle are used. These methods handle instances that occlude each other better since they aim at solving an optimal assignment of the votes to competing hypotheses.

The backprojection can also be used to obtain a link between a hypothesis and the training data [31]. For instance, Equation (24) can be modified by taking only offsets $D_c^{L_t(\boldsymbol{y})}(\theta)$ into account that were sampled from a specific training

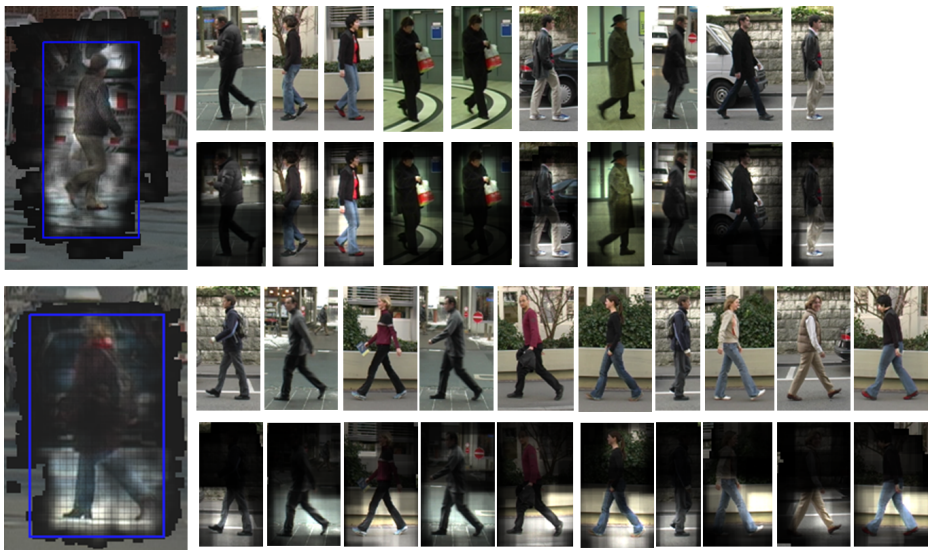


Fig. 9. Two object hypotheses and their top ten nearest training examples (ordered from left to right). The detected pedestrians are the same as in Figure 8. For each hypothesis, the top row shows the training examples that contribute most to the hypothesis. The bottom row shows the backprojection mask superimposed on each training example. The images have been taken from [31].

example θ. The contribution of a training example for a hypothesis is then measured by $\sum_{\boldsymbol{y}} \pi(\boldsymbol{h}, \boldsymbol{y}, \theta)$. Figure 9 shows the training examples that contribute most to the detections shown in Figure 8.

More general, the similarity between two hypotheses $\boldsymbol{h}_1$ and $\boldsymbol{h}_2$ of the same class c can be defined by

$$S(\boldsymbol{h}_1, \boldsymbol{h}_2) = \frac{\sum_t \sum_{L_t} \sum_{\boldsymbol{d} \in D_c^{L_t}} \frac{p(c|L_t)}{|D_c^{L_t}|} \mathbb{I}(\boldsymbol{d}, \boldsymbol{h}_1) \mathbb{I}(\boldsymbol{d}, \boldsymbol{h}_2)}{\sum_t \sum_{L_t} \sum_{\boldsymbol{d} \in D_c^{L_t}} \frac{p(c|L_t)}{|D_c^{L_t}|} \mathbb{I}(\boldsymbol{d}, \boldsymbol{h}_1)} \tag{25}$$

where

$$\mathbb{I}(\boldsymbol{d}, \boldsymbol{h}) = \begin{cases} 0 & \text{if } \max_{\boldsymbol{y}} \pi(\boldsymbol{h}, \boldsymbol{y}, \boldsymbol{d}) = 0, \\ 1 & \text{otherwise.} \end{cases} \tag{26}$$

The indicator function $\mathbb{I}(\boldsymbol{d}, \boldsymbol{h})$ is basically 1 if an offset $\boldsymbol{d}$ contributes to a hypothesis $\boldsymbol{h}$, which is measured by $\pi(\boldsymbol{h}, \boldsymbol{y}, \boldsymbol{d})$, i.e., Equation (24) computed for a single offset $\boldsymbol{d}$ instead of $\sum_{\boldsymbol{d} \in D_c^{L_t(\boldsymbol{y})}}$.

Having a similarity measure, one can retrieve the nearest neighbors from the training set and transfer attributes from them to the detection hypothesis. For instance, the viewpoint of a detected car is estimated using Equation (25) in [31]. The most interesting property of the similarity measure based on the support of two hypotheses is the robustness to occlusions as shown in Figure 10.

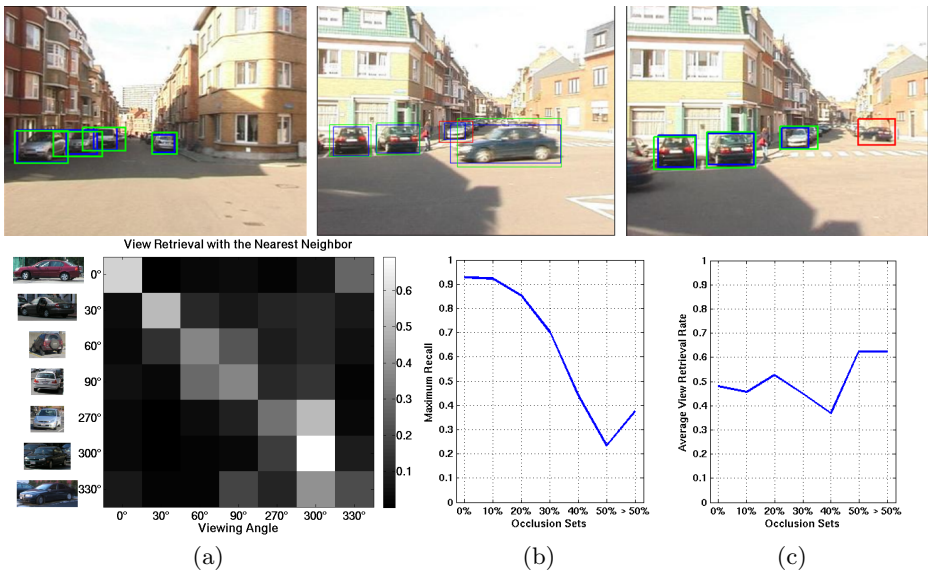


Fig. 10. Viewpoint retrieval on the Leuven car dataset [21]; some examples are shown in the top row (blue - ground truth, green - correct detection, red - incorrect detection). (a) Confusion matrix. Most of the confusions appear between neighboring viewpoints. (b-c) The viewpoint retrieval performance with respect to the amount of occlusion. Although the detection performance deteriorates with an increasing amount of occlusion (b), the viewpoint retrieval performance is affected very little (c), which shows the robustness of the similarity measure to occlusions. The images have been taken from [31].

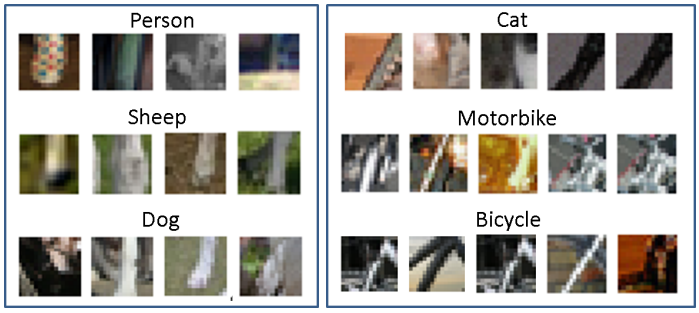
3.6 Feature Sharing

The advantage of using one multi-class detector compared to having a detector for each positive class is the ability of sharing features among classes, which reduces the memory requirements and also the testing time. The sharing and the performance of a random forest for multi-class object detection on the PASCAL VOC 2006 and 2007 datasets [11] have been investigated in [32].

The sharing among classes is illustrated in Figure 11. Since each leaf contains patches from several classes, one can compute the amount of sharing among classes [32] by

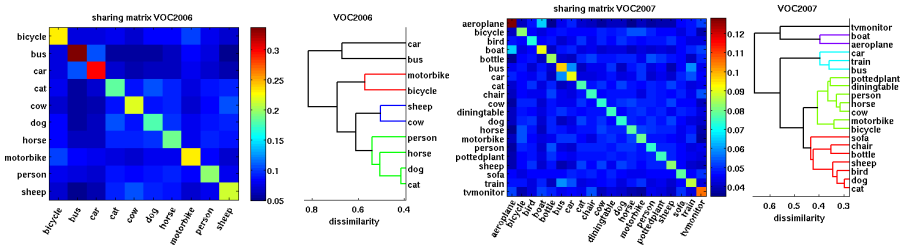
$$T(c_i, c_j) \propto \sum_t \sum_{L_t} \left(\left| D_{c_i}^{L_t} \right| \cdot p\left(c_j | L_t\right) \right), \tag{27}$$

where $T(c_i, c_j)$ is normalized such that $\sum_j T(c_i, c_j) = 1$. The obtained sharing matrix T among the positive classes for the two datasets PASCAL VOC 2006 and 2007 are shown in Figure 12. For PASCAL VOC 2006, many features are shared between the pairs bus-car, cat-dog, motorbike-bicycle, and cow-sheep since these categories are also similar in appearance and shape. For dissimilar categories like bus and cow, the sharing is marginal.



(a) (b)

Fig. 11. Patches clustered in two leaves of a multi-class detector trained on PASCAL VOC 2006. The first leaf shares features of similar appearance among the classes person, sheep, and dog. The second example shares features among the classes cat, motorbike, and bicycle. The images have been taken from [32].



(a) PASCAL VOC 2006 (b) PASCAL VOC 2007

Fig. 12. Sharing matrices and their corresponding taxonomies which are automatically obtained by clustering the sharing matrices. The images have been taken from [32].

Based on the sharing matrix T, one can derive a taxonomy of classes by clustering the symmetric dissimilarity matrix $D = 1 - \frac{1}{2}(T + T^T)$ using complete-linkage agglomerative clustering. The automatically derived taxonomies are also plotted in Figure 12. The taxonomies show that the feature sharing within a multi-class random forest is meaningful. The close similarity between cow and sheep can be explained by the typical green background of the training images of the two classes. Since the forest is trained on bounding boxes, many patches with class labels for cow and sheep contain mainly grass from the background. The sharing, however, depends on the image features that are used for the splitting functions. For instance, potted plant and dinning table are measured as similar for PASCAL VOC 2007. Since potted plants are not well described by the used histogram of gradients features, the location of the category in the taxonomy is not very meaningful.

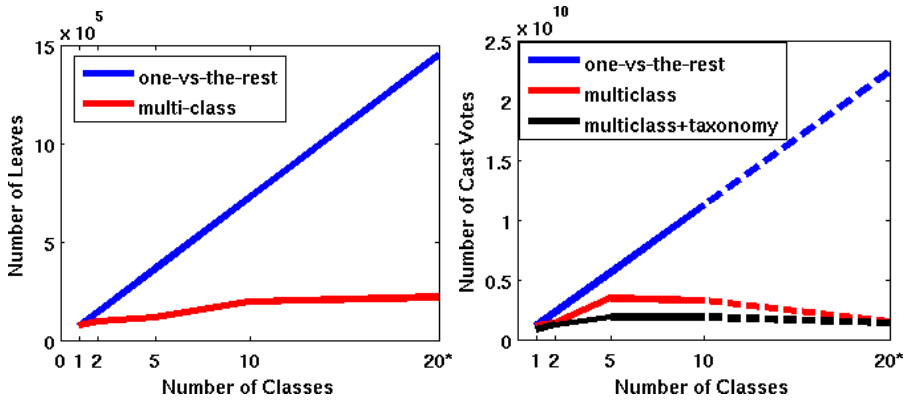


Fig. 13. Both the number of leaves (a) and the number of votes (b) of a multi-class random forest grow sublinearly with respect to the number of classes. In contrast, one-vs-the-rest approaches grow linearly. The difference between the blue and the red curve in (a) indicates the amount of sharing that is happening. In (b), the derived taxonomy can be used to further reduce the number of votes. The images have been taken from [32].

Figure 13 shows the effect of sharing of a multi-class random forest in comparison to training a random forest for each positive class (one-vs-the-rest). Not only the number of leaves is reduced, yielding less memory requirements, but also the votes to be cast for detection is lower. This is achieved by casting only votes if $p\left(c_j|L_t\right) > \frac{1}{C}$, where C is the number of classes. Based on the taxonomy, one can even adjust the thresholds for the classes to reduce the number of cast votes further [32].

The multi-class forest can also be used to generate class hypotheses that are verified with a more sophisticated classifier or detector. In [32], the verification detector [14] has been used for re-scoring each hypothesis. The performance on PASCAL VOC 2006 is shown in Table 1. The multi-class random forest (MC) and the taxonomy (T) perform similar or better than many one-vs-the-rest (OvA) random forests even after the verification step. While the number of verifications scales well with the number of classes as shown in Table 2, there is no loss in performance compared to [14]. As reported in [32], the system requires 35 seconds per image for detecting one positive class, but only 100 seconds for detecting all 20 classes. Comparing these numbers with the fast verification detector [14], which requires 7 seconds per image and per class and 134 seconds per image for 20 classes, there is already a benefit for less than 20 classes. Although it is clear that 100 seconds are still not satisfying, optimizing the random forest for multi-class object detection as in [18] or using the approximations mentioned in this chapter might give a significant reduction of the detection time.

Method	bic.	bus	car	cat	cow	dog	hrs.	m.bi.	pers.	shp.	avg
OvA	.16	.13	.07	.04	.18	.03	.15	.16	.11	.12	.114
MC	.37	.12	.11	.02	.14	.05	.08	.21	.05	.12	.127
MC+T.	.38	.13	.12	.05	.15	.03	.11	.12	.05	.12	.132
[14]	.64	.62	.634	.23	.46	.14	.45	.61	.38	.45	.459
OvA+vrf.	.67	.62	.62	.23	.46	.14	.46	.62	.35	.43	.461
MC+vrf.	.68	.64	.65	.20	.47	.14	.44	.64	.38	.43	.465
MC+T.+vrf.	.66	.64	.66	.22	.47	.14	.44	.64	.36	.42	.463

Table 1. Performance comparison of a multi-class method (MC) with some baselines in average-precision for the PASCAL VOC 2006 dataset. The first block shows the detection without verification and without non-maxima suppression. MC outperforms one-vs-the-rest (OvA). The taxonomy not only reduces the amount of voting (Figure 13), it also gives a slight improvement. In the second block, verification is performed with [14]. By using a two-stage method, there is no loss in accuracy compared to [14]. The number of performed verifications is given in Table 2.

Method	#windows	#verifications
MC-VOC'06 (10 cat.)	1321	1321
MC-VOC'07 (20 cat.)	1778	1778
[14]-VOC'07(20 cat.)	42278	833141

Table 2. The multi-class random forest (MC) reduces the number of windows for verification per image. Since the hypotheses already have a class label, each hypothesis or window needs to be verified only once. It is important that the reduction is achieved without compromising accuracy; see Table 1.

4 Discussion and Conclusion

In this chapter, we have described a general random forest framework for multi-class object detection and discussed several implementation variations. In this context, object detection is formulated as a combined regression and classification problem. While the detection problem becomes a distribution estimation problem, the random forests allow to learn features and descriptors that are optimal for estimating the distributions with low uncertainty. The theoretical framework, however, has the shortcoming that general distributions become too expensive for large datasets. Therefore, several approximations have been discussed to improve the efficiency. The approximations range from restricting the type of distributions to Gaussians or Gaussian mixture models to using an approximation of the spatial distribution for measuring the gain or using subsampling strategies during training and testing. Although many approximations are very intuitive and the basic algorithm is straightforward to implement, it requires some engineering to find an optimal trade-off between accuracy and runtime performance. The most crucial parameter for the detection accuracy, however, is the amount of training data. Random forests are not designed to generalize from small training sets, but to handle large amount of training data efficiently. For datasets with limited training data and large intra-class variation

like PASCAL VOC 2007, they do not achieve the best detection accuracy without an additional verification step [17, 32]. However, using semi-supervised learning and regularizers that exploit large amount of unlabeled data as described in this chapter might overcome the overfitting problem of random forests partially. Due to its relation to implicit shape models [22], the detection approach shares advantages and limitations of this type of models. While techniques like back-projection and feature sharing allow to reason about object hypotheses and the similarity of categories, which goes beyond black box classifiers, the independent assumption of the image patches is a weakness of these models that needs to be addressed in the future. Nevertheless, random forests have a strong potential for applications where many labeled examples are available. For instance, pose or body part estimation from depth data [12, 18] are examples where accurate results can be obtained in real-time. The work [18] also shows the benefit of engineering where a fine tuned version of [17] resulted in a speed-up by a factor of 3200.

References

1. Amit, Y., Geman, D.: Shape quantization and recognition with randomized trees. Neural Computation 9(7), 1545–1588 (1997)
2. Barinova, O., Lempitsky, V., Kohli, P.: On the detection of multiple object instances using hough transforms. In: IEEE Conf. Computer Vision and Pattern Recognition (2010)
3. Bay, H., Ess, A., Tuytelaars, T., Van Gool, L.: Speeded-up robust features (surf). Computer Vision and Image Understanding 110(3), 346 – 359 (2008)
4. Bosch, A., Zisserman, A., Muñoz, X.: Image classification using random forests and ferns. In: Int'l Conf. Computer Vision (2007)
5. Breiman, L.: Random forests. Machine Learning 45(1), 5–32 (2001)
6. Chen, H., Liu, T., Fuh, C.: Segmenting highly articulated video objects with weak-prior randomforests. In: European Conf. Computer Vision. pp. 373–385 (2006)
7. Criminisi, A., Shotton, J., Konukoglu, E.: Decision forests for classification, regression, density estimation, manifold learning and semi-supervised learning. Tech. Rep. MSR-TR-2011-114, Microsoft Research, Cambridge (2011)
8. Criminisi, A., Shotton, J., Robertson, D., Konukoglu, E.: Regression forests for efficient anatomy detection and localization in ct studies. In: Medical Computer Vision Workshop (2010)
9. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: IEEE Conf. Computer Vision and Pattern Recognition. pp. 886–893 (2005)
10. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: IEEE Conf. Computer Vision and Pattern Recognition (2009)
11. Dollár, P., Wojek, C., Schiele, B., Perona, P.: Pedestrian detection: An evaluation of the state of the art. IEEE Trans. Pattern Analysis and Machine Intelligence (2012)
12. Everingham, M., Van Gool, L., Williams, C., Winn, J., Zisserman, A.: The pascal visual object classes (voc) challenge. International Journal of Computer Vision 88(2), 303–338 (2010)

13. Fanelli, G., Gall, J., Van Gool, L.: Real time head pose estimation with random regression forests. In: IEEE Conf. Computer Vision and Pattern Recognition (2011)
14. Fanelli, G., Weise, T., Gall, J., Van Gool, L.: Real time head pose estimation from consumer depth cameras. In: Pattern Recognition, pp. 101–110 (2011)
15. Felzenszwalb, P., Girshick, R., McAllester, D., Ramanan, D.: Object detection with discriminatively trained part based models. IEEE Trans. Pattern Analysis and Machine Intelligence 32, 1627–1645 (2010)
16. Gall, J., Lempitsky, V.: Class-specific hough forests for object detection. In: IEEE Conf. Computer Vision and Pattern Recognition (2009)
17. Gall, J., Razavi, N., Van Gool, L.: On-line adaption of class-specific codebooks for instance tracking. In: British Machine Vision Conf. (2010)
18. Gall, J., Yao, A., Razavi, N., Van Gool, L., Lempitsky, V.: Hough forests for object detection, tracking, and action recognition. IEEE Trans. Pattern Analysis and Machine Intelligence 33, 2188–2202 (2011)
19. Girshick, R., Shotton, J., Kohli, P., Criminisi, A., Fitzgibbon, A.: Efficient regression of general-activity human poses from depth images. Int'l Conf. Computer Vision (2011)
20. Godec, M., Roth, P., Bischof, H.: Hough-based tracking of non-rigid objects. In: Int'l Conf. Computer Vision (2011)
21. Lehmann, A., Leibe, B., Van Gool, L.: Fast prism: Branch and bound hough transform for object class detection. Int'l J. Computer Vision 94, 175–197 (2011)
22. Leibe, B., Cornelis, N., Cornelis, K., Van Gool, L.: Dynamic 3d scene analysis from a moving vehicle. In: IEEE Conf. Computer Vision and Pattern Recognition (2007)
23. Leibe, B., Leonardis, A., Schiele, B.: Robust object detection with interleaved categorization and segmentation. Int'l J. Computer Vision 77(1-3), 259–289 (2008)
24. Leistner, C., Godec, M., Schulter, S., Saffari, A., Werlberger, M., Bischof, H.: Improving classifiers with unlabeled weakly-related videos. In: IEEE Conf. Computer Vision and Pattern Recognition (2011)
25. Leistner, C., Saffari, A., Santner, J., Bischof, H.: Semi-supervised random forests. In: Int'l Conf. Computer Vision. pp. 506–513 (2009)
26. Lepetit, V., Lagger, P., Fua, P.: Randomized trees for real-time keypoint recognition. In: IEEE Conf. Computer Vision and Pattern Recognition. pp. 775–781 (2005)
27. Lowe, D.: Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision 60(2), 91 (2004)
28. Marée, R., Geurts, P., Piater, J., Wehenkel, L.: Random subwindows for robust image classification. In: IEEE Conf. Computer Vision and Pattern Recognition. pp. 34–40 (2005)
29. Menze, B., Kelm, B., Splitthoff, D., Koethe, U., Hamprecht, F.: On oblique random forests. In: Machine Learning and Knowledge Discovery in Databases, pp. 453–469 (2011)
30. Moosmann, F., Triggs, B., Jurie, F.: Fast discriminative visual codebooks using randomized clustering forests. In: Neural Information Processing Systems (2006)
31. Okada, R.: Discriminative generalized hough transform for object dectection. In: Int'l Conf. Computer Vision (2009)
32. Razavi, N., Gall, J., Van Gool, L.: Backprojection revisited: Scalable multi-view object detection and similarity metrics for detections. In: European Conf. Computer Vision (2010)
33. Razavi, N., Gall, J., Van Gool, L.: Scalable multi-class object detection. In: IEEE Conf. Computer Vision and Pattern Recognition. pp. 1505–1512 (2011)

34. Rematas, K., Leibe, B.: Efficient object detection and segmentation with a cascaded hough forest ism. In: IEEE Workshop on Challenges and Opportunities in Robot Perception (2011)
35. Schroff, F., Criminisi, A., Zisserman, A.: Object class segmentation using random forests. In: British Machine Vision Conf. (2008)
36. Schulter, S., Leistner, C., Roth, P., Bischof, H., Van Gool, L.: On-line hough forests. In: British Machine Vision Conf. (2011)
37. Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., Blake, A.: Real-time human pose recognition in parts from single depth images. In: IEEE Conf. Computer Vision and Pattern Recognition (2011)
38. Shotton, J., Johnson, M., Cipolla, R.: Semantic texton forests for image categorization and segmentation. In: IEEE Conf. Computer Vision and Pattern Recognition (2008)
39. Sun, M., Bradski, G., Xu, B.X., Savarese, S.: Depth-encoded hough voting for coherent object detection, pose estimation, and shape recovery. In: European Conf. Computer Vision (2010)
40. Winn, J., Shotton, J.: The layout consistent random field for recognizing and segmenting partially occluded objects. In: IEEE Conf. Computer Vision and Pattern Recognition. pp. 37–44 (2006)