Fast Edge Detection Using Structured Forests

Piotr Dollár and C. Lawrence Zitnick
Microsoft Research
{pdollar,larryz}@microsoft.com

Abstract—Edge detection is a critical component of many vision systems, including object detectors and image segmentation algorithms. Patches of edges exhibit well-known forms of local structure, such as straight lines or T-junctions. In this paper we take advantage of the structure present in local image patches to learn both an accurate and computationally efficient edge detector. We formulate the problem of predicting local edge masks in a structured learning framework applied to random decision forests. Our novel approach to learning decision trees robustly maps the structured labels to a discrete space on which standard information gain measures may be evaluated. The result is an approach that obtains realtime performance that is orders of magnitude faster than many competing state-of-the-art approaches, while also achieving state-of-the-art edge detection results on the BSDS500 Segmentation dataset and NYU Depth dataset. Finally, we show the potential of our approach as a general purpose edge detector by showing our learned edge models generalize well across datasets.

1 Introduction

Edge detection has remained a fundamental task in computer vision since the early 1970's [18], [15], [43]. The detection of edges is a critical preprocessing step for a variety of tasks, including object recognition [47], [17], segmentation [33], [1], and active contours [26]. Traditional approaches to edge detection use a variety of methods for computing color gradients followed by non-maximal suppression [7], [19], [50]. Unfortunately, many visually salient edges do not correspond to color gradients, such as texture edges [34] and illusory contours [39]. State-of-the-art edge detectors [1], [41], [31], [21] use multiple features as input, including brightness, color, texture and depth gradients computed over multiple scales.

Since visually salient edges correspond to a variety of visual phenomena, finding a unified approach to edge detection is difficult. Motivated by this observation several recent papers have explored the use of learning techniques for edge detection [13], [49], [31], [27]. These approaches take an image patch and compute the likelihood that the center pixel contains an edge. Optionally, the independent edge predictions may then be combined using global reasoning [1], [41], [49], [2].

Edges in a local patch are highly interdependent [31]. They often contain well-known patterns, such as straight lines, parallel lines, T-junctions or Y-junctions [40], [31]. Recently, a family of learning approaches called *structured learning* [36] has been applied to problems exhibiting similar characteristics. For instance, [29] applies structured learning to the problem of semantic image labeling for which local image labels are also highly interdependent.

In this paper we propose a generalized structured learning approach that we apply to edge detection. This approach allows us to take advantage of the inherent structure in edge patches, while being surprisingly computationally efficient. We can compute edge maps in realtime, which is orders of magnitude faster than competing state-of-the-art approaches. A random forest framework is used to capture the structured information [29]. We formulate the problem of edge detection as predicting local segmentation masks given input image patches. Our novel approach to learning decision trees uses structured labels to determine the splitting function at each branch in the tree. The structured labels are robustly mapped to a discrete space on which standard information gain measures may be evaluated. Each forest predicts a patch of edge pixel labels that are aggregated across the image to compute our final edge map, see Figure 1. Since the aggregated edge maps may be diffuse, the edge maps may optionally be sharpened using local color and depth cues. We show state-of-the-art results on both the BSDS500 [1] and the NYU Depth dataset [44]. We demonstrate the potential of our approach as a general purpose edge detector by showing the strong cross dataset generalization of our learned edge models.

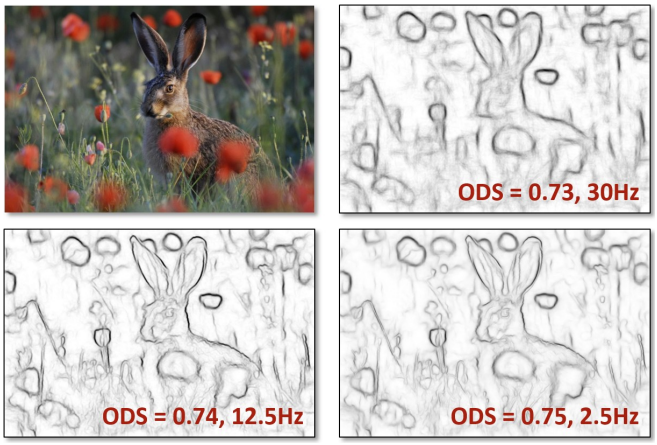

Fig. 1. Edge detection results using three versions of our Structured Edge (SE) detector demonstrating tradeoffs in accuracy vs. runtime. We obtain realtime performance while simultaneously achieving state-of-the-art results. ODS numbers were computed on BSDS [1] on which the popular gPb detector [1] achieves a score of .73. The variants shown include SE, SE+SH, and SE+MS+SH, see §4 for details.

1.1 Related work

We now discuss related work in edge detection and structured learning. An earlier version of this work appeared in [14].

Edge detection: Numerous papers have been written on edge detection over the past 50 years. Early work [18], [15], [7], [37], [19] focused on the detection of intensity or color gradients. The popular Canny detector [7] finds the peak gradient orthogonal to edge direction. An evaluation of various low-level edge detectors can be found in [4] and an overview in [50]. More recent work [34], [32], [28], [1], [48], [30] explores edge detection under more challenging conditions.

Several techniques have explored the use of learning for edge detection [13], [49], [32], [41], [31], [27]. Dollár et al. [13] used a boosted classifier to independently label each pixel using its surrounding image patch as input. Zheng et al. [49] combine low, mid, and high-level cues and show improved results for object-specific edge detection. Ren and Bo [41] improved the result of [1] by computing gradients across learned sparse codes of patch gradients. While [41] achieved good results, their approach further increased the high computational cost of [1]. Catanzaro et al. [8] improve the runtime of [1] using parallel algorithms. Recently, Kivinen et al. [27] applied deep networks to edge detection achieving competitive results.

Finally, Lim et al. [31] propose an edge detection approach that classifies edge patches into *sketch tokens* using random forest classifiers, that, like in our work, attempt to capture local edge structure. Sketch tokens bear resemblance to earlier work on *shapemes* [40] but are computed directly from color image patches rather than from pre-computed edge maps. The result is an efficient approach for detecting edges that also shows promising results for object detection. In contrast to previous work, we do not require the use of pre-defined classes of edge patches. This allows us to learn more subtle variations in edge structure and leads to a more accurate and efficient algorithm.

Structured learning: Structured learning addresses the problem of learning a mapping where the input or output space may be arbitrarily complex representing strings, sequences, graphs, object pose, bounding boxes etc. [46], [45], [3]. We refer readers to [36] for a comprehensive survey.

Our structured random forests differ from these works in several respects. First, we assume that the output space is structured but operate on a standard input space. Second, by default our model can only output examples observed during training, which implicitly assumes the existence of a set of representative samples (this shortcoming can be ameliorated with custom ensemble models). On the other hand, typical structured predictors learn parameters to a scoring function and at inference perform an optimization to obtain predictions [46], [36]. This requires defining a scoring function and an efficient (possibly approximate) inference procedure. In contrast, inference using our structured random forest is straightforward, general and fast (same as for standard random forests).

Finally, our work was inspired by recent work from Kontschieder et al. [29] on learning random forests for structured class labels for the specific case where the output labels represent a semantic image labeling for an image patch. The key observation made by Kontschieder et al. is that given a color image patch, the leaf node reached in a tree is independent of the structured semantic labels, and any type of output can be stored at each leaf. Building on this, we propose a general learning framework for structured output forests that can be used with a broad class of output spaces. We apply our framework to learning an accurate and fast edge detector.

2 Random Decision Forests

We begin with a review of random decision forests [6], [5], [20]. Throughout our presentation we adopt the notation and terminology of the extensive recent survey by Criminisi et al. [11], somewhat simplified for ease of presentation. The notation in [11] is sufficiently general to support our extension to random forests with structured outputs.

A decision tree $f_t(x)$ classifies a sample $x \in \mathcal{X}$ by recursively branching left or right down the tree until a leaf node is reached. Specifically, each node j in the tree is associated with a binary *split function*:

$$h(x, \theta_j) \in \{0, 1\} \tag{1}$$

with parameters θ_j. If $h(x, \theta_j) = 0$ node j sends x left, otherwise right, with the process terminating at a leaf node. The output of the tree on an input x is the prediction stored at the leaf reached by x, which may be a target label $y \in \mathcal{Y}$ or a distribution over the labels $\mathcal{Y}$.

While the split function $h(x, \theta)$ may be arbitrarily complex, a common choice is a 'stump' where a single feature dimension of x is compared to a threshold. Specifically, $\theta = (k, \tau)$ and $h(x, \theta) = [x(k) < \tau]$, where $[\cdot]$ denotes the indicator function. Another popular choice is $\theta = (k_1, k_2, \tau)$ and $h(x, \theta) = [x(k_1) - x(k_2) < \tau]$. Both are computationally efficient and effective in practice [11].

A decision forest is an ensemble of T independent trees f_t. Given a sample x, the predictions $f_t(x)$ from the set of trees are combined using an *ensemble model* into a single output. Choice of ensemble model is problem specific and depends on $\mathcal{Y}$, common choices include majority voting for classification and averaging for regression, although more sophisticated ensemble models may be employed [11].

Observe that arbitrary information may be stored at the leaves of a decision tree. The leaf node reached by the tree depends only on the input x, and while predictions of multiple trees must be merged in some useful way (the ensemble model), any type of output y can be stored at each leaf. This allows use of complex output spaces $\mathcal{Y}$, including structured outputs as observed by Kontschieder et al. [29].

While prediction is straightforward, training random decision forests with structured $\mathcal{Y}$ is more challenging. We review the standard learning procedure next and describe our generalization to learning with structured outputs in §3.

2.1 Training Decision Trees

Each tree is trained independently in a recursive manner. For a given node j and training set $\mathcal{S}_j \subset \mathcal{X} \times \mathcal{Y}$, the goal is to find parameters θ_j of the split function $h(x, \theta_j)$ that result in a 'good' split of the data. This requires defining an *information gain criterion* of the form:

$$I_j = I(\mathcal{S}_j, \mathcal{S}_j^L, \mathcal{S}_j^R) \tag{2}$$

where $\mathcal{S}_j^L = \{(x,y) \in \mathcal{S}_j | h(x,\theta_j) = 0\}$, $\mathcal{S}_j^R = \mathcal{S}_j \backslash \mathcal{S}_j^L$. Splitting parameters θ_j are chosen to maximize the information gain I_j; training then proceeds recursively on the left node with data $\mathcal{S}_j^L$ and similarly for the right node. Training stops when a maximum depth is reached or if information gain or training set size fall below fixed thresholds.

For multiclass classification ($\mathcal{Y} \subset \mathbb{Z}$) the standard definition of information gain can be used:

$$I_j = H(\mathcal{S}_j) - \sum_{k \in \{L,R\}} \frac{|S_j^k|}{|S_j|} H(S_j^k) \tag{3}$$

where $H(\mathcal{S}) = -\sum_y p_y \log(p_y)$ denotes the Shannon entropy and p_y is the fraction of elements in $\mathcal{S}$ with label y. Alternatively the Gini impurity $H(\mathcal{S}) = \sum_y p_y(1-p_y)$ has also been used in conjunction with Eqn. (3) [6].

For regression, entropy and information gain can be extended to continuous variables [11]. Alternatively, a common approach for single-variate regression ($\mathcal{Y} = \mathbb{R}$) is to minimize the variance of labels at the leaves [6]. If we write the variance as $H(S) = \frac{1}{|\mathcal{S}|}\sum_y (y-\mu)^2$ where $\mu = \frac{1}{|\mathcal{S}|}\sum_y y$, then substituting H for entropy in Eqn. (3) leads to the standard criterion for single-variate regression.

Can we define a more general *information gain* criterion for Eqn. (2) that generalizes well for arbitrary output spaces $\mathcal{Y}$? Surprisingly yes, given mild additional assumptions about $\mathcal{Y}$. Before going into detail in §3, we discuss the key role that randomness plays in the training of decision forests next.

2.2 Randomness and Optimality

Individual decision trees exhibit high variance and tend to overfit [24], [6], [5], [20]. Decision forests ameliorate this by training multiple de-correlated trees and combining their output. A crucial component of the training procedure is therefore to achieve a sufficient diversity of trees.

Diversity of trees can be obtained either by randomly subsampling the data used to train each *tree* [6] or randomly subsampling the features and splits used to train each *node* [24]. Injecting randomness at the level of nodes tends to produce higher accuracy models [20] and has proven more popular [11]. Specifically, when optimizing Eqn. (2), only a small set of possible θ_j are sampled and tested when choosing the optimal split. E.g., for stumps where $\theta = (k,\tau)$ and $h(x,\theta) = [x(k) < \tau]$, [20] advocates sampling $\sqrt{d}$ features where $\mathcal{X} = \mathbb{R}^d$ and a single threshold τ per feature.

In effect, accuracy of individual trees is sacrificed in favor of a high diversity ensemble [20]. Leveraging similar intuition allows us to introduce an approximate information gain criterion for structured labels, described next, and leads to our generalized structured forest formulation.

3 Structured Random Forests

In this section we extend random decision forests to general structured output spaces $\mathcal{Y}$. Of particular interest for computer vision is the case where $x \in \mathcal{X}$ represents an image patch and $y \in \mathcal{Y}$ encodes the corresponding local image annotation (e.g., a segmentation mask or set of semantic image labels). However, we keep our derivation general.

Training random forests with structured labels poses two main challenges. First, structured output spaces are often high dimensional and complex. Thus scoring numerous candidate splits directly over structured labels may be prohibitively expensive. Second, and more critically, information gain over structured labels may not be well defined.

We use the observation that even *approximate* measures of information gain suffice to train effective random forest classifiers [20], [29]. 'Optimal' splits are not necessary or even desired, see §2.2. Our core idea is to map all the structured labels $y \in \mathcal{Y}$ at a given node into a discrete set of labels $c \in \mathcal{C}$, where $\mathcal{C} = \{1,\ldots,k\}$, such that *similar* structured labels y are assigned to the same discrete label c.

Given the discrete labels $\mathcal{C}$, information gain calculated directly and efficiently over $\mathcal{C}$ can serve as a proxy for the information gain over the structured labels $\mathcal{Y}$. As a result at each node we can leverage existing random forest training procedures to learn structured random forests effectively.

Our approach to calculating information gain relies on measuring similarity over $\mathcal{Y}$. However, for many structured output spaces, including those used for edge detection, computing similarity over $\mathcal{Y}$ is not well defined. Instead, we define a mapping of $\mathcal{Y}$ to an *intermediate* space $\mathcal{Z}$ in which distance is easily measured. We therefore utilize a broadly applicable two-stage approach of first mapping $\mathcal{Y} \to \mathcal{Z}$ followed by a straightforward mapping of $\mathcal{Z} \to \mathcal{C}$.

We describe the proposed approach in more detail next and return to its application to edge detection in §4.

3.1 Intermediate Mapping Π

Our key assumption is that for many structured output spaces, including for structured learning of edge detection, we can define a mapping of the form:

$$\Pi : \mathcal{Y} \to \mathcal{Z} \tag{4}$$

such that we can approximate dissimilarity of $y \in \mathcal{Y}$ by computing Euclidean distance in $\mathcal{Z}$. For example, as we describe in detail in §4, for edge detection the labels $y \in \mathcal{Y}$ are 16×16 segmentation masks and we define $z = \Pi(y)$ to be a long binary vector that encodes whether every pair of pixels in y belong to the same or different segments. Distance is easily measured in the resulting space $\mathcal{Z}$.

$\mathcal{Z}$ may be high dimensional which presents a challenge computationally. For example, for edge detection there are $\binom{16\cdot16}{2} = 32640$ unique pixel pairs in a 16×16 segmentation mask, so computing z for every y would be expensive. However, as only an approximate distance measure is necessary, the dimensionality of $\mathcal{Z}$ can be reduced.

In order to reduce dimensionality, we sample m dimensions of $\mathcal{Z}$, resulting in a reduced mapping $\Pi_\phi : \mathcal{Y} \to \mathcal{Z}$ parametrized by ϕ. During training, a distinct mapping Π_ϕ is randomly generated and applied to training labels $\mathcal{Y}_j$ at each node j. This serves two purposes. First, Π_ϕ can be considerably faster to compute than Π. Second, sampling $\mathcal{Z}$

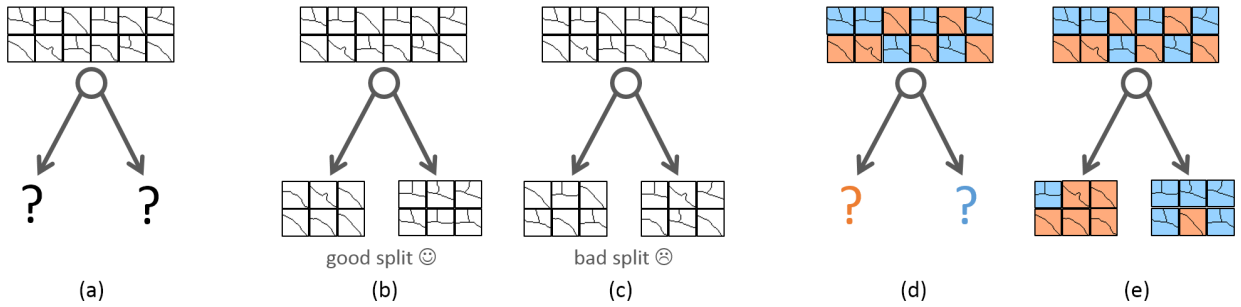

Fig. 2. Illustration of the decision tree node splits: (a) Given a set of structured labels such as segments, a splitting function must be determined. Intuitively a good split (b) groups similar segments, whereas a bad split (c) does not. In practice we cluster the structured labels into two classes (d). Given the class labels, a standard splitting criterion, such as Gini impurity, may be used (e).

injects additional randomness into the learning process and helps ensure a sufficient diversity of trees, see §2.2.

Finally, Principal Component Analysis (PCA) [25] can be used to further reduce the dimensionality of $\mathcal{Z}$. PCA denoises $\mathcal{Z}$ while approximately preserving Euclidean distance. In practice, we use Π_ϕ with $m = 256$ dimensions followed by a PCA projection to at most 5 dimensions.

3.2 Information Gain Criterion

Given the mapping $\Pi_\phi : \mathcal{Y} \to \mathcal{Z}$, a number of choices for the information gain criterion are possible. For discrete $\mathcal{Z}$ multivariate joint entropy could be computed directly. Kontschieder et al. [29] proposed such an approach, but due to its complexity of $O(|\mathcal{Z}|^m)$, were limited to using $m \leq 2$. Our experiments indicate $m \geq 64$ is necessary to accurately capture similarities between elements in $\mathcal{Z}$. Alternatively, given continuous $\mathcal{Z}$, variance or a continuous formulation of entropy [11] can be used to define information gain. In this work we propose a simpler, extremely efficient approach.

We map a set of structured labels $y \in \mathcal{Y}$ into a discrete set of labels $c \in \mathcal{C}$, where $\mathcal{C} = \{1, \ldots, k\}$, such that labels with similar z are assigned to the same discrete label c, see Figure 2. The discrete labels may be binary ($k = 2$) or multiclass ($k > 2$). This allows us to use standard information gain criteria based on Shannon entropy or Gini impurity as defined in Eqn. (3). Critically, discretization is performed independently when training each node and depends on the distribution of labels at a given node (contrast with [31]).

We consider two straightforward approaches to obtaining the discrete label set $\mathcal{C}$ given $\mathcal{Z}$. Our first approach is to cluster z into k clusters using K-means (projecting z onto 5 dimensions prior to clustering). Alternatively, we can quantize z based on the top $\log_2(k)$ PCA dimensions, assigning z a discrete label c according to the orthant (generalization of quadrant) into which z falls. Both approaches perform similarly but the latter is slightly faster. We use PCA quantization to obtain $k = 2$ labels unless otherwise specified.

3.3 Ensemble Model

Finally, we define how to combine a set of n labels $y_1 \ldots y_n$ into a single prediction for both training (to set leaf labels) and testing (to merge predictions). As before, we sample an m dimensional mapping Π_ϕ and compute $z_i = \Pi_\phi(y_i)$ for each i. We select the label y_k whose z_k is the medoid, i.e. the z_k that minimizes the sum of distances to all other z_i[1]. Note that typically we only need to compute the medoid for small n (either for training a leaf node or merging the output of multiple trees), hence using a coarse distance metric suffices.

The biggest limitation is that any prediction $y \in \mathcal{Y}$ must have been observed during training; the ensemble model is unable to synthesize novel labels. Indeed, this is impossible without additional information about $\mathcal{Y}$. In practice, domain specific ensemble models are preferable. For example, in edge detection we apply structured prediction to obtain edge maps for each image patch independently and merge overlapping predictions by averaging (note that in this case structured prediction operates at the patch level and not the image level).

4 Edge Detection

We now describe how to apply our structured forests to edge detection. As input our method takes an image that may contain multiple channels, such as an RGB or RGBD image. The task is to label each pixel with a binary variable indicating whether the pixel contains an edge or not. Similar to the task of semantic image labeling [29], the labels within a small image patch are highly interdependent, providing a promising candidate problem for our structured forest approach.

We assume we are given a set of segmented training images, in which the boundaries between the segments correspond to contours [1], [44]. Given an image patch, its annotation can be specified either as a *segmentation mask* indicating segment membership for each pixel (defined up to a permutation) or a binary *edge map*. We use $y \in \mathcal{Y} = \mathbb{Z}^{d \times d}$ to denote the former and $y' \in \mathcal{Y}' = \{0, 1\}^{d \times d}$ for the latter, where d indicates patch width. An edge map y' can always be trivially derived from segmentation mask y, but not vice versa. We utilize both representations in our approach.

Next, we describe how we compute the input features x, the mapping functions Π_ϕ used to determine splits, and the ensemble model used to combine multiple predictions.

Input features: Our learning approach predicts a structured 16×16 segmentation mask from a larger 32×32 image patch. We begin by augmenting each image patch with multiple additional *channels* of information, resulting in a feature vector

1. The medoid z_k minimizes $\sum_{ij}(z_{kj} - z_{ij})^2$. This is equivalent to $\min_k \sum_j (z_{kj} - \bar{z}_j)^2$ and can be computed efficiently in time $O(nm)$.

$x \in \mathbb{R}^{32\times32\times K}$ where K is the number of channels. We use features of two types: pixel lookups $x(i,j,k)$ and pairwise differences $x(i_1,j_1,k) - x(i_2,j_2,k)$, see §2.

Inspired by Lim et al. [31], we use a similar set of color and gradient channels (originally developed for fast pedestrian detection [12]). We compute 3 color channels in CIE-LUV color space along with normalized gradient magnitude at 2 scales (original and half resolution). Additionally, we split each gradient magnitude channel into 4 channels based on orientation. The result is 3 color, 2 magnitude and 8 orientation channels, for a total of 13 channels.

We blur the channels with a radius 2 triangle filter and downsample by a factor of 2, resulting in $32\cdot32\cdot13/4 = 3328$ candidate features x. Motivated by [31], we also compute pairwise difference features. We apply a large triangle blur to each channel (8 pixel radius), and downsample to a resolution of 5×5. Sampling all candidate pairs and computing their differences yields an additional $\binom{5\cdot5}{2} = 300$ candidate features per channel, resulting in 7228 total candidate features.

Mapping function: To train decision trees, we need to define a mapping $\Pi : \mathcal{Y} \to \mathcal{Z}$ as described in §3. Recall that our structured labels y are 16×16 segmentation masks. One option is to use $\Pi : \mathcal{Y} \to \mathcal{Y}'$, where y' represents the binary edge map corresponding to y. Unfortunately Euclidean distance over $\mathcal{Y}'$ yields a brittle distance measure.

We therefore define an alternate mapping Π. Let $y(j)$ for $1 \le j \le 256$ denote the segment index of the j^{th} pixel of y. Individually a single value $y(j)$ yields no information about y, since y is defined only up to a permutation. Instead we can sample a pair of locations $j_1 \neq j_2$ and check if they belong to the same segment, $y(j_1) = y(j_2)$. This allows us to define $z = \Pi(y)$ as a large binary vector that encodes $[y(j_1) = y(j_2)]$ for every unique pair of indices $j_1 \neq j_2$. While $\mathcal{Z}$ has $\binom{256}{2}$ dimensions, in practice we only compute a subset of m dimensions as discussed in §3.2. We found a setting of $m = 256$ and $k = 2$ gives good results, effectively capturing the similarity of segmentation masks.

Ensemble model: Random forests achieve robust results by combining the output of multiple trees. While merging segmentation masks $y \in \mathcal{Y}$ for overlapping patches is difficult, multiple overlapping edge maps $y' \in \mathcal{Y}'$ can be averaged to yield a soft edge response. Thus in addition to the learned mask y, we also store the corresponding edge map y' at each leaf node, thus allowing predictions to be combined quickly and simply through averaging during inference.

Efficiency: The surprising efficiency of our approach derives from the use of structured labels that predict information for an entire image neighborhood. This greatly reduces the number of trees T that need to be evaluated. We compute our structured output densely on the image with a stride of 2 pixels, thus with 16×16 output patches, each pixel receives $16^2T/4 \approx 64T$ predictions. In practice we use $T = 4$ and thus the score of each pixel in the output edge map is averaged over 256 votes.

A critical assumption is that predictions are uncorrelated. Since both the inputs and outputs of each tree overlap, we train $2T$ total trees and evaluate an alternating set of T trees at each adjacent location. Use of such a 'checkerboard pattern' improves results somewhat, introducing larger separation between the trees did not improve results further.

4.1 Multiscale Detection (SE+MS)

We now describe the first of two enhancements to our base algorithm. Inspired by the work of Ren [38], we implement a multiscale version of our edge detector. Given an input image I, we run our structured edge detector on the original, half, and double resolution version of I and average the result of the three edge maps after resizing to the original image dimensions. Although somewhat inefficient, the approach noticeably improves edge quality. We refer to the multiscale version of our structured edge detector as SE+MS.

4.2 Edge Sharpening (SE+SH)

We observed that predicted edge maps from our structured edge detector are somewhat diffuse. For strong, isolated edges non-maximal suppression can be used to effectively detect edge peaks. However, given fine image structures edge responses can 'bleed' together resulting in missed detections; likewise, for weak edges that receive few votes no clear peak may emerge. The underlying cause for the diffuse edge responses is that the individually predicted edge maps are noisy and are not perfectly aligned to each other or the underlying image data. Specifically, each overlapping prediction may be shifted by a few pixels from the true edge location.

To address this phenomenon, we introduce a new *sharpening* procedure that aligns edge responses from overlapping predictions. Our core observation is that local image color and depth values can be used to more precisely localize predicted responses. Intuitively, given a predicted segmentation mask, the mask can be morphed slightly so that it better matches the underlying image patch. Aligning overlapping masks to the underlying image data implicitly aligns the masks with each other, resulting in sharper, better localized edge responses.

Sharpening takes a predicted segmentation mask $y \in \mathcal{Y}$ and the corresponding image patch $x \in \mathcal{X}$ and produces a new mask that better aligns to x. As before, let $y(j)$ denote the segment index of the j^{th} pixel of mask y. First, for each segment s, we compute its mean color $\mu_s = E[x(j)|y(j) = s]$ using all pixels j in s. Next, we iteratively update the assigned segment for each pixel by assigning it to the segment which minimizes $\|\mu_s - x(j)\|_2$. For each pixel we restrict the set of assignable segments to the segments that are immediately adjacent to it (4-connected neighborhood). Given the new sharpened segmentation masks, we compute and average their corresponding edge maps as before. However, since the edge maps are better aligned to the image data the resulting aggregated edge map is sharper.

Sharpening can be repeated multiple times prior to averaging the corresponding edge maps. Experiments reveal that the first sharpening step produces the largest gains, and in practice two steps suffice. Taking advantage of the sparsity of edges, the sharpening procedure can be implemented efficiently. For details we direct readers to source code. Note that sharpening is not guaranteed to improve results but we find it is quite effective in practice. We refer to the sharpened versions of our structured edge detector as SE+SH and SE+MS+SH.

Fig. 3. Illustration of edge detection results on the BSDS500 dataset on five sample images. The first two rows show the original image and ground truth. The next three rows contain results for gPb-owt-ucm [1], Sketch Tokens [31], and SCG [41]. The final four rows show our results for variants of SE. Use viewer zoom functionality to see fine details.

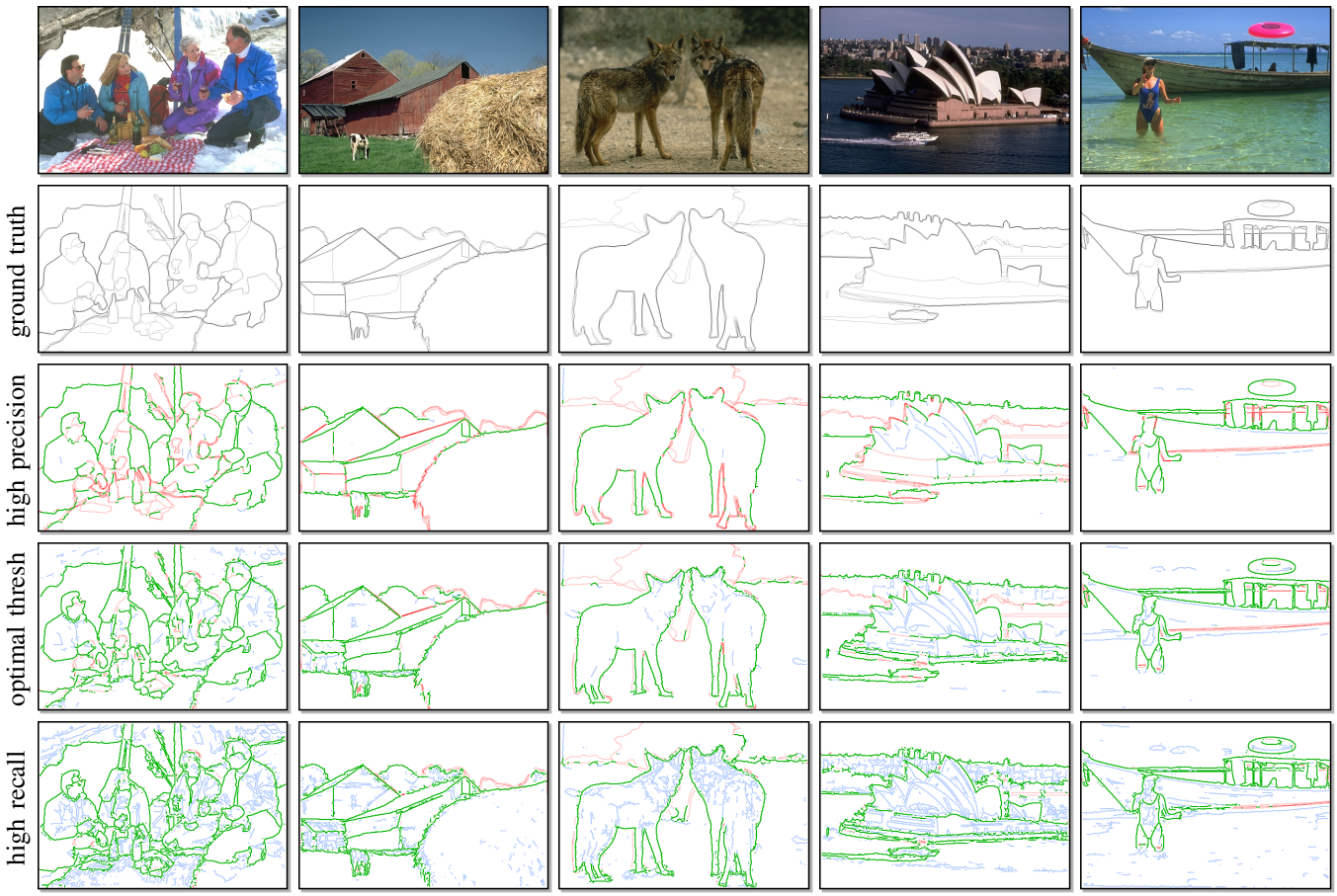

Fig. 4. Visualizations of matches and errors of SE+MS+SH compared to BSDS ground truth edges. Edges are thickened to two pixels for better visibility; the color coding is green=true positive, blue=false positive, red=false negative. Results are shown at three thresholds: high precision (T≈.26, P≈0.88, R=.50), ODS threshold (T≈.14, P=R≈.75), and high recall (T≈.05, P=.50, R≈0.93).

5 Results

In this section we analyze the performance of our structured edge (SE) detector in detail. First we analyze the influence of parameters in §5.1 and test SE variants in §5.2. Next, we compare results on the BSDS [1] and NYUD [44] datasets to the state-of-the-art in §5.3 and §5.4, respectively, reporting both accuracy and runtime. We conclude by demonstrating the cross dataset generalization of our approach in §5.5.

The majority of our experiments are performed on the Berkeley Segmentation Dataset and Benchmark (BSDS500) [35], [1]. The dataset contains 200 training, 100 validation, and 200 testing images. Each image has hand labeled ground truth contours. Edge detection accuracy is evaluated using three standard measures: fixed contour threshold (ODS), per-image best threshold (OIS), and average precision (AP) [1]. To evaluate accuracy in the high recall regime, we additionally introduce a new measure, recall at 50% precision (R50), in §5.2. Prior to evaluation, we apply a standard non-maximal suppression technique to our edge maps to obtain thinned edges [7]. Example detections on BSDS are shown in Figure 3 and visualizations of edge accuracy are shown in Figure 4.

5.1 Parameter Sweeps

We set all parameters with the help of the BSDS validation set which is fully independent of the test set. Parameters include: structured forest splitting parameters (e.g., m and k), feature parameters (e.g., image and channel blurring), and model and tree parameters (e.g. number of trees and data quantity). Training takes ~20 minute per tree using one million patches and is parallelized over trees. Evaluation of trees is parallelized as well, we use a quad-core machine for all reported runtimes.

In Figures 5-7 we explore the effect of choices of **splitting**, **model** and **feature** parameters. For each experiment we train on the 200 image training set and measure edge detection accuracy on the 100 image validation set (using the standard ODS performance metric). All results are averaged over 5 trials. First, we set all parameters to their default values indicated by orange markers in the plots. Then, keeping all but one parameter fixed, we explore the effect on edge detection accuracy as a single parameter is varied.

Since we explore a large number of parameters settings, we perform our experiments using a slightly reduced accuracy model that is faster to train. Specifically we train using fewer patches ($2 \cdot 10^5$ versus 10^6) and utilize sharpening (SH) but not multiscale detection (MS). Also, the validation set is more challenging than the test set and we evaluate using 25 thresholds instead of 99, further reducing accuracy (.71 ODS). Finally, we note that sweep details have changed slightly from the our previous work [14]; most notably, the sweeps now utilize sharpening but not multiscale detection.

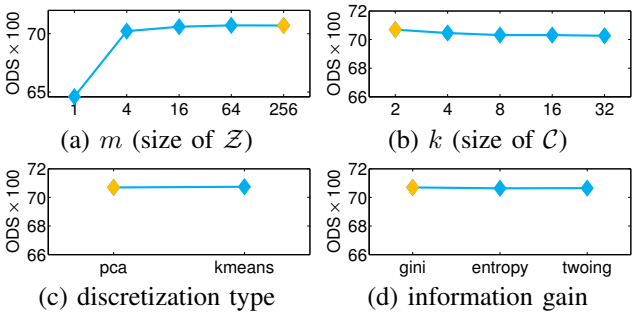

Fig. 5. Splitting parameter sweeps. See text for details.

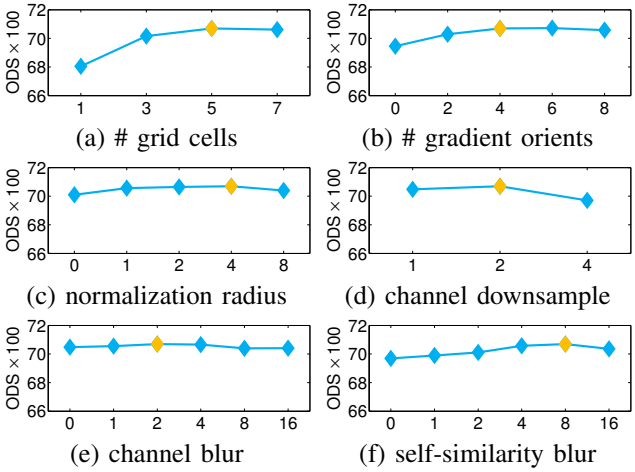

Fig. 6. Feature parameter sweeps. See text for details.

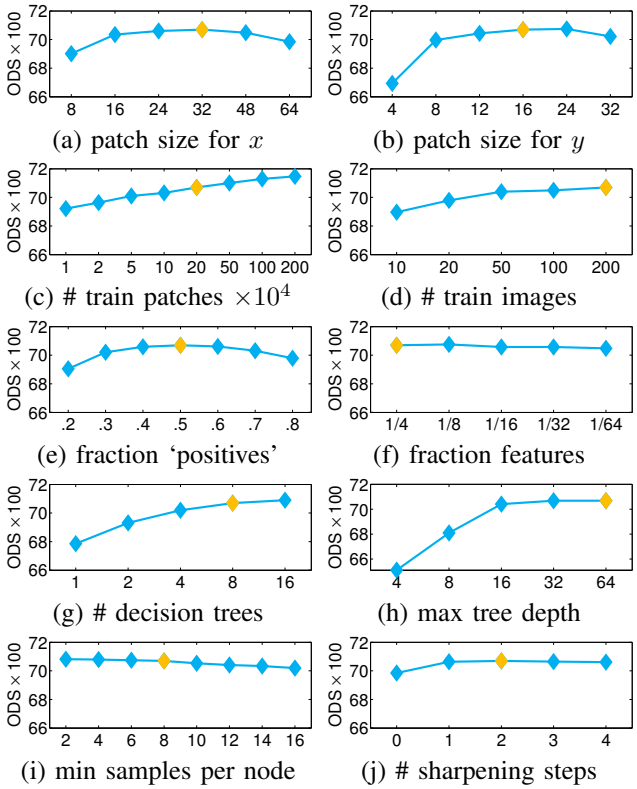

Fig. 7. Model parameter sweeps. See text for details.

Splitting Parameters: In Figure 5 we explore how best to measure information gain over structured labels. Recall we utilize a two-stage approach of mapping $\mathcal{Y} \rightarrow \mathcal{Z}$ followed by $\mathcal{Z} \rightarrow \mathcal{C}$. Plots (a) and (b) demonstrate that $m = |\mathcal{Z}|$ should be large and $k = |\mathcal{C}|$ small. Results are robust to both the discretization method and the discrete measure of information gain as shown in plots (c) and (d).

Feature Parameters: Figure 6 shows how varying the channel features affects accuracy. We refer readers to §4 and source code for details, here we only note that performance is relatively insensitive to a broad range of parameter settings.

Model Parameters: In Figure 7 we plot the influence of parameters governing the model and training data. (a) and (b) show the effect of image and label patch sizes on accuracy, 32×32 image patches and 16×16 label patches are best. (c) and (d) show that increasing the number of patches and training images improves accuracy. (e) shows that about half the sampled patches should be 'positive' (have more than one ground truth segment) and (f) shows that training each tree with a fraction of total features has negligible impact on accuracy (but results in proportionally lower memory usage). In (g)-(i) we see that many, deep, un-pruned trees give best performance (nevertheless, we prune trees so every node has at least 8 training samples to decrease model size). Finally (j) shows that two sharpening steps give best results. Impact of sharpening is explored in more detail next in §5.2.

5.2 Structured Edge Variants

Given our trained detector, sharpening (SH) and multiscale detection (MS) can be used to enhance results. In this section we analyze the performance of the four resulting combinations. We emphasize that the same trained model can be used with sharpening and multiscale detection enabled at runtime.

In Figure 8 we plot precision/recall curves for the four variants of our approach: SE, SE+MS, SE+SH, and SE+MS+SH. Summary statistics are reported in the bottom rows of Table 1. SE has an ODS score of .73 and SE+MS and SE+SH both achieve an ODS of .74. SE+MS+SH, which combines multiscale detection and sharpening[2], yields an ODS of .75. In all cases OIS, which is measured using a separate optimal threshold per image, is about 2 points higher than ODS.

As these results indicate, both sharpening and multiscale detection improve accuracy. To further analyze these enhancements, we introduce a new evaluation metric: recall at 50% precision (R50), which measures accuracy in the high recall regime. SE achieves R50 of .90 and SE+MS does not improve this. In contrast, SE+SH boosts R50 considerably to .93. This increase in recall for the SE variants can clearly be seen in Figure 8. The MS variants, on the other hand, improve precision in the low-recall regime. Overall, both SH and MS improve AP[3], with SE+SH+MS achieving an AP of .80.

2. We trained the top-performing SE+MS+SH model with $4 \cdot 10^6$ patches compared to 10^6 for the other SE models, further increasing ODS by $\sim$.004.

3. We discovered an error in the BSDS evaluation which overestimates AP by 1%. For consistency with past results, however, we used the code as is.

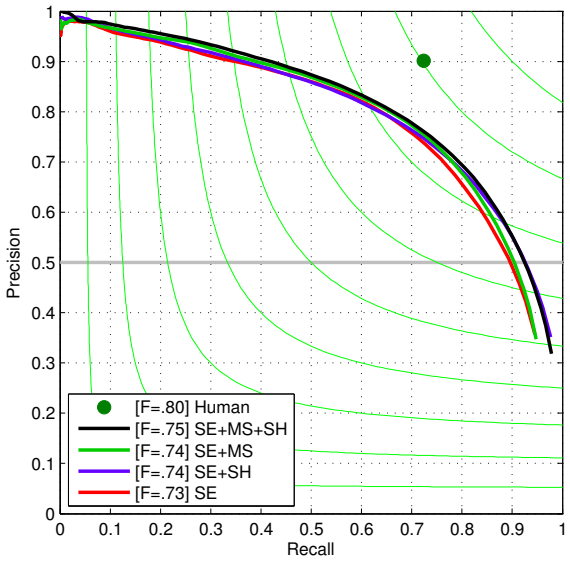

Fig. 8. Results of structured edges (SE) with sharpening (+SH) and multiscale detection (+MS). SH increases recall while MS increases precision; their combination gives best results.

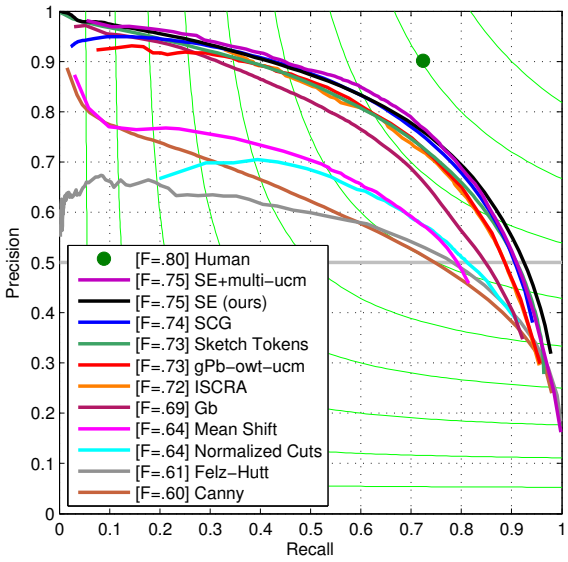

Fig. 9. Results on BSDS500. Structured edges (SE) and SE coupled with hierarchical multiscale segmentation (SE+multi-ucm) [2] achieve top results. For the SE result we report the SE+MS+SH variant. See Table 1 for additional details including method citations and runtimes. SE is orders of magnitude faster than nearly all edge detectors with comparable accuracy.

	ODS	OIS	AP	R50	FPS
Human	.80	.80	-	-	-
Canny	.60	.63	.58	.75	15
Felz-Hutt [16]	.61	.64	.56	.78	10
Normalized Cuts [10]	.64	.68	.45	.81	-
Mean Shift [9]	.64	.68	.56	.79	-
Hidayat-Green [23]	.62†	-	-	-	20
BEL [13]	.66†	-	-	-	1/10
Gb [30]	.69	.72	.72	.85	1/6
gPb + GPU [8]	.70†	-	-	-	1/2‡
ISCRA [42]	.72	.75	.46	.89	1/30‡
gPb-owt-ucm [1]	.73	.76	.73	.89	1/240
Sketch Tokens [31]	.73	.75	.78	.91	1
DeepNet [27]	.74	.76	.76	-	1/5‡
SCG [41]	.74	.76	.77	.91	1/280
SE+multi-ucm [2]	**.75**	**.78**	.76	.91	1/15
SE	.73	.75	.77	.90	**30**
SE+SH	.74	.76	.79	**.93**	12.5
SE+MS	.74	.76	.78	.90	6
SE+MS+SH	**.75**	.77	**.80**	**.93**	2.5

TABLE 1
Results on BSDS500. †BSDS300 results. ‡Utilizes the GPU.

The runtime of the four variants is reported in the last column of Table 1. SE runs at a frame rate of 30hz, enabling real time processing. Both SH and MS slow the detector, with MS incurring a higher cost. Nevertheless, SE+SH runs at over 12hz while achieving excellent accuracy. Indeed, in the high recall regime, which is necessary for many common scenarios, SE+SH achieves top results. Given its speed and high recall, we expect SE+SH to be the default variant used in practice.

5.3 BSDS500 Results

We compare our edge detector against competing methods, reporting both accuracy and runtime. Precision/recall curves are shown in Figure 9 and summary statistics are in Table 1.

Our full approach, SE+MS+SH, outperforms all state-of-the-art approaches [41], [27], [31], [1], [42]. We improve ODS/OIS by 1 point over competing methods and AP/R50 by 2 points. Our edge detector is particularly effective in the high recall regime. The only method with comparable accuracy is SE+multi-ucm [2] which couples our SE+MS detector with a hierarchical multiscale segmentation approach.

SE+MS+SH is orders of magnitude faster than nearly all edge detectors with comparable accuracy, see last column of Table 1. All runtimes are reported on 480×320 images. Our approach scales linearly with image size and and is parallelized across four cores. While many competing methods are likewise linear, they have a much higher cost per pixel. The single scale variant of our detector, SE+SH, further improves speed by $5\times$ with only minor loss in accuracy and no loss in recall. Finally, SE runs at 30hz while still achieving competitive accuracy.

In comparison to other learning-based approaches to edge detection, we considerably outperform BEL [13] which computes edges independently at each pixel given its surrounding image patch. We also outperform Sketch Tokens [31] in both accuracy and runtime performance. This may be the result of Sketch Tokens using a fixed set of classes for selecting split criterion at each node, whereas our structured forests can capture finer patch edge structure. Moreover, our structured output leads to significantly smoother edge maps, see Figure 3. Finally, Kivinen et al. [27] recently trained deep networks for edge detection; unfortunately, we were unable to obtain results from the authors to perform detailed comparisons.

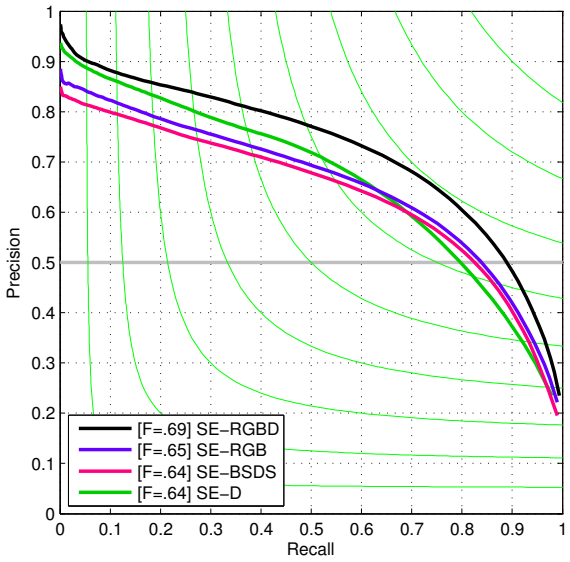

Fig. 10. Precision/recall curves on NYUD using different image modalities. SE-BSDS is the RGB model trained on the BSDS dataset. See Table 2 and 3 and text for details.

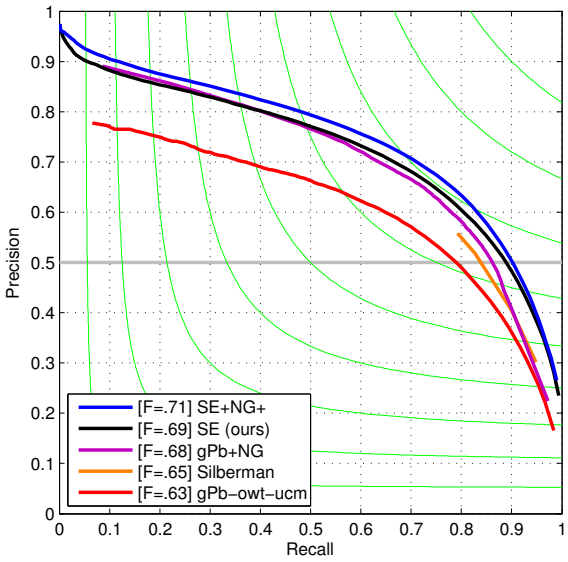

Fig. 11. Results on NYUD. Structured edges (SE) and SE coupled with depth normal gradient (SE+NG+) [21], [22] achieve top results. For the SE result we report the SE+SH variant. See Table 2 for additional details including citations and runtimes.

	ODS	OIS	AP	R50	FPS
gPb-owt-ucm [1]	.63	.66	.56	.79	1/360
Silberman [44]	.65	.66	.29	.84	1/360+
gPb+NG [21]	.68	.71	.63	.86	1/375
SE+NG+ [22]	**.71**	**.72**	**.74**	**.90**	1/15
SE-D	.64	.65	.66	.80	**7.5**
SE-RGB	.65	.67	.65	.84	**7.5**
SE-RGBD	.69	.71	.72	.89	5

TABLE 2
Results on the NYUD dataset [44].

5.4 NYUD Results

The NYU Depth (NYUD) dataset [44] is composed of 1449 pairs of RGB and depth images with corresponding semantic segmentations. The dataset was adopted independently for edge detection by Ren and Bo [41] and by Gupta et al. [21]. In practice, the two dataset variants have significant differences. Specifically, [41] proposed a different train/test split and used half resolution images. Instead, [21] use the train/test split originally proposed by Silberman et al. [44] and full resolution images. In our previous work we used the version from [41] in our experiments; in the present work we switch to the version from Gupta et al. [21] as more methods have been evaluated on this variant and it utilizes the full resolution images.

Gupta et al. [21] split the NYUD dataset into 381 training, 414 validation, and 654 testing images and generated ground truth edges from the semantic segmentations provided by [44]. The original 640×480 images are cropped to discard boundary regions with missing ground truth. Finally the maximum slop allowed for correct matches of edges to ground truth during evaluation is increased from .0075 of the image diagonal to .011. This is necessary to compensate for the relatively inexact localization of the ground truth.

Example SE results are shown in Figure 12. We treat the depth channel in the same manner as the other color channels. Specifically, we recompute the gradient channels over the depth channel (with identical parameters) resulting in 11 additional channels. Precision/recall curves for SE+SH with different image modalities are shown in Figure 10. Use of depth information only (SE-D) gives good precision as strong depth discontinuities nearly always correspond to edges. Use of intensity information only (SE-RGB) gives better recall as nearly all edges have intensity discontinuities but not all edges have depth discontinuities. As expected, simultaneous use of intensity and depth (SE-RGBD) substantially improves results.

Summary statistics are given in Table 2. Runtime is slower than on BSDS as NYUD images are higher resolution and features must be computed over both intensity and depth. For these results we utilized the SE+SH variant which slightly outperformed SE+MS+SH on this dataset.

In Table 2 and Figure 11 we compare our approach, SE+SH, to a number of state-of-the-art approaches, including gPb-owt-ucm (color only), Silberman et al.'s RGBD segmentation algorithm [44], and detectors from Gupta et al. [21], [22] that explicitly estimate depth normal gradient (NG). While our approach naively utilizes depth (treating the depth image identically to the intensity image), we outperform nearly all competing methods, including gPb+NG [21]. Gupta et al. [22] obtain top results by coupling our structured edges detector with depth normal gradients and additional cues (SE+NG+). Finally, for a comparison of SE to Ren and Bo's SCG [41] on the alternate version of NYUD we refer readers to our previous work [14] (in the alternate setup SE likewise outperforms SCG across all modalities while retaining its speed advantage).

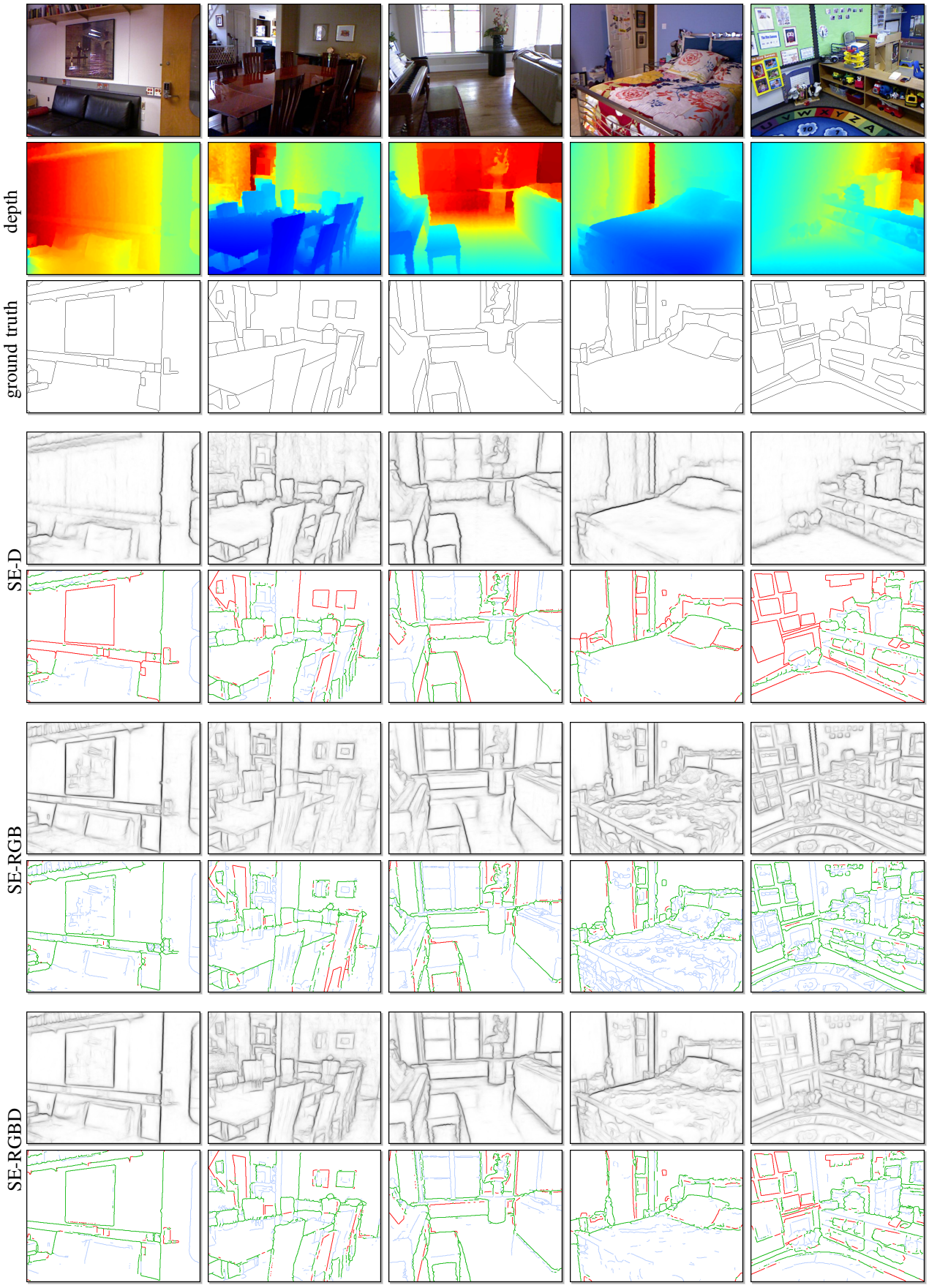

Fig. 12. Edge detection results on the NYUD dataset. Edges from depth features (SE-D) have good precision, edges from intensity features (SE-RGB) give better recall, and simultaneous use of intensity and depth (SE-RGBD) gives best results. For details about visualizations of matches and errors see Figure 4; all visualizations were generated using each detector's ODS threshold.

	ODS	OIS	AP	R50	FPS
NYUD / NYUD	.65	.67	.65	.84	7.5
BSDS / NYUD	.64	.66	.63	.83	7.5
BSDS / BSDS	.75	.77	.80	.93	2.5
NYUD / BSDS	.73	.74	.77	.91	2.5

TABLE 3
Cross-dataset generalization for Structured Edges. TRAIN/TEST indicates the training/testing dataset used.

5.5 Cross dataset generalization

To study the ability of our approach to generalize across datasets we ran a final set of experiments. In Table 3 we show results on NYUD using structured forests trained on BSDS and also results on BSDS using structured forests trained on NYUD. For these experiments we use intensity images only. Note that images in the BSDS and NYUD datasets are qualitatively quite different, see Figure 3 and 12, respectively.

Table 3, top, compares results on NYUD of the NYUD and BSDS trained models. Across all performance measure, scores degrade by about 1 point when using the BSDS dataset for training. Precision/recall curves on NYUD for the NYUD model (SE-RGB) and the BSDS model (SE-BSDS) are shown in Figure 10. The resulting curves align closely. The minor performance change is surprising given the different statistics of the datasets. Results on BSDS of the BSDS and NYUD models, shown in Table 3, bottom, are likewise similar.

These experiments provide strong evidence that our approach could serve as a general purpose edge detector without the necessity of retraining. We expect this to be a critical aspect of our detector allowing for its widespread applicability.

6 Discussion

Our approach is capable of realtime frame rates while achieving state-of-the-art accuracy. This may enable new applications that require high-quality edge detection and efficiency. For instance, our approach may be well suited for video segmentation or for time sensitive object recognition tasks such as pedestrian detection.

Our approach to learning structured decision trees may be applied to a variety of problems. The fast and direct inference procedure is ideal for applications requiring computational efficiency. Given that many vision applications contain structured data, there is significant potential for structured forests in other applications.

In conclusion, we propose a structured learning approach to edge detection. We describe a general purpose method for learning structured random decision forest that robustly uses structured labels to select splits in the trees. We demonstrate state-of-the-art accuracies on two edge detection datasets, while being orders of magnitude faster than most competing state-of-the-art methods.

Source code is available online.

References

[1] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *PAMI*, 33, 2011. 1, 2, 4, 6, 7, 9, 10
[2] P. Arbeláez, J. Pont-Tuset, J. T. Barron, F. Marques, and J. Malik. Multiscale combinatorial grouping. In *CVPR*, 2014. 1, 9
[3] M. Blaschko and C. Lampert. Learning to localize objects with structured output regression. In *ECCV*, 2008. 2
[4] K. Bowyer, C. Kranenburg, and S. Dougherty. Edge detector evaluation using empirical roc curves. *Computer Vision and Image Understanding*, 84(1):77–103, 2001. 2
[5] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct. 2001. 2, 3
[6] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and Regression Trees*. Chapman and Hall/CRC, 1984. 2, 3
[7] J. Canny. A computational approach to edge detection. *PAMI*, 8(6):679–698, November 1986. 1, 2, 7
[8] B. Catanzaro, B.-Y. Su, N. Sundaram, Y. Lee, M. Murphy, and K. Keutzer. Efficient, high-quality image contour detection. In *ICCV*, 2009. 2, 9
[9] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *PAMI*, 24:603–619, 2002. 9
[10] T. Cour, F. Benezit, and J. Shi. Spectral segmentation with multiscale graph decomposition. In *CVPR*, 2005. 9
[11] A. Criminisi, J. Shotton, and E. Konukoglu. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and Trends in Computer Graphics and Vision*, 7(2-3), February 2012. 2, 3, 4
[12] P. Dollár, S. Belongie, and P. Perona. The fastest pedestrian detector in the west. In *BMVC*, 2010. 5
[13] P. Dollár, Z. Tu, and S. Belongie. Supervised learning of edges and object boundaries. In *CVPR*, 2006. 1, 2, 9
[14] P. Dollár and C. L. Zitnick. Structured forests for fast edge detection. In *ICCV*, 2013. 2, 7, 10
[15] R. O. Duda, P. E. Hart, et al. *Pattern classification and scene analysis*, volume 3. Wiley New York, 1973. 1, 2
[16] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *IJCV*, 59(2):167–181, 2004. 9
[17] V. Ferrari, L. Fevrier, F. Jurie, and C. Schmid. Groups of adjacent contour segments for object detection. *PAMI*, 30(1):36–51, 2008. 1
[18] J. R. Fram and E. S. Deutsch. On the quantitative evaluation of edge detection schemes and their comparison with human performance. *IEEE TOC*, 100(6), 1975. 1, 2
[19] W. T. Freeman and E. H. Adelson. The design and use of steerable filters. *PAMI*, 13:891–906, 1991. 1, 2
[20] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learn*, 63(1):3–42, Apr. 2006. 2, 3
[21] S. Gupta, P. Arbelaez, and J. Malik. Perceptual organization and recognition of indoor scenes from RGB-D images. In *CVPR*, 2013. 1, 10
[22] S. Gupta, R. Girshick, P. Arbelaez, and J. Malik. Learning rich features from RGB-D images for object detection and segmentation. In *ECCV*, 2014. 10
[23] R. Hidayat and R. Green. Real-time texture boundary detection from ridges in the standard deviation space. In *BMVC*, 2009. 9
[24] T. K. Ho. The random subspace method for constructing decision forests. *PAMI*, 20(8):832–844, 1998. 3
[25] I. T. Joliffe. *Principal Component Analysis*. Springer-Verlag, 1986. 4
[26] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *IJCV*, 1(4):321–331, 1988. 1
[27] J. J. Kivinen, C. K. Williams, and N. Heess. Visual boundary prediction: A deep neural prediction network and quality dissection. In *AISTATS*, 2014. 1, 2, 9
[28] I. Kokkinos. Boundary detection using f-measure-, filter- and feature- (F^3) boost. In *ECCV*, 2010. 2
[29] P. Kontschieder, S. Bulo, H. Bischof, and M. Pelillo. Structured class-labels in random forests for semantic image labelling. In *ICCV*, 2011. 1, 2, 3, 4
[30] M. Leordeanu, R. Sukthankar, and C. Sminchisescu. Generalized boundaries from multiple image interpretations. *PAMI*, 2014. 2, 9
[31] J. Lim, C. L. Zitnick, and P. Dollár. Sketch tokens: A learned mid-level representation for contour and object detection. In *CVPR*, 2013. 1, 2, 4, 5, 6, 9
[32] J. Mairal, M. Leordeanu, F. Bach, M. Hebert, and J. Ponce. Discriminative sparse image models for class-specific edge detection and image interpretation. In *ECCV*, 2008. 2
[33] J. Malik, S. Belongie, T. Leung, and J. Shi. Contour and texture analysis for image segmentation. *IJCV*, 43, 2001. 1
[34] D. Martin, C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *PAMI*, 26(5):530–549, 2004. 1, 2

[35] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *ICCV*, 2001. 7
[36] S. Nowozin and C. H. Lampert. Structured learning and prediction in computer vision. *Foundations and Trends in Computer Graphics and Vision*, 6:185–365, 2011. 1, 2
[37] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *PAMI*, 12(7):629–639, 1990. 2
[38] X. Ren. Multi-scale improves boundary detection in natural images. In *ICCV*, 2008. 5
[39] X. Ren, C. Fowlkes, and J. Malik. Scale-invariant contour completion using cond. random fields. In *ICCV*, 2005. 1
[40] X. Ren, C. Fowlkes, and J. Malik. Figure/ground assignment in natural images. In *ECCV*, 2006. 1, 2
[41] X. Ren and B. Liefeng. Discriminatively trained sparse code gradients for contour detection. In *NIPS*, 2012. 1, 2, 6, 9, 10
[42] Z. Ren and G. Shakhnarovich. Image segmentation by cascaded region agglomeration. In *CVPR*, 2013. 9
[43] G. S. Robinson. Color edge detection. *Optical Engineering*, 16(5), 1977. 1
[44] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012. 1, 4, 7, 10
[45] B. Taskar, V. Chatalbashev, D. Koller, and C. Guestrin. Learning structured prediction models: a large margin approach. In *ICML*, 2005. 2
[46] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Learning for interdependent and structured output spaces. In *ICML*, 2004. 2
[47] S. Ullman and R. Basri. Recognition by linear combinations of models. *PAMI*, 13(10), 1991. 1
[48] N. Widynski and M. Mignotte. A particle filter framework for contour detection. In *ECCV*, 2012. 2
[49] S. Zheng, Z. Tu, and A. Yuille. Detecting object boundaries using low-, mid-, and high-level information. In *CVPR*, 2007. 1, 2
[50] D. Ziou, S. Tabbone, et al. Edge detection techniques-an overview. *Pattern Recognition and Image Analysis*, 8:537–559, 1998. 1, 2

Piotr Dollár received his masters in computer science from Harvard University in 2002 and his PhD from the University of California, San Diego in 2007. He joined the Computational Vision lab at Caltech as a postdoctoral fellow in 2007. Upon being promoted to senior postdoctoral fellow he realized it time to move on, and in 2011, he joined Microsoft Research. In 2014 he became a member of Facebook AI Research (FAIR), where he currently resides. He has worked on object detection, pose estimation, boundary learning and behavior recognition. His general interests lie in machine learning and pattern recognition and their application to computer vision.

C. Lawrence Zitnick is a senior researcher in the Interactive Visual Media group at Microsoft Research, and is an affiliate associate professor at the University of Washington. He is interested in a broad range of topics related to visual object recognition. His current interests include object detection and semantically interpreting visual scenes. He developed the PhotoDNA technology used by Microsoft, Facebook and various law enforcement agencies to combat illegal imagery on the web. Previous research topics include computational photography, stereo vision, and image-based rendering. Before joining MSR, he received the PhD degree in robotics from Carnegie Mellon University in 2003. In 1996, he co-invented one of the first commercial portable depth cameras.