Holistically-Nested Edge Detection

Saining Xie
Dept. of CSE and Dept. of CogSci
University of California, San Diego
9500 Gilman Drive, La Jolla, CA 92093
s9xie@eng.ucsd.edu

Zhuowen Tu
Dept. of CogSci and Dept. of CSE
University of California, San Diego
9500 Gilman Drive, La Jolla, CA 92093
ztu@ucsd.edu

Abstract

We develop a new edge detection algorithm that tackles two important issues in this long-standing vision problem: (1) holistic image training and prediction; and (2) multi-scale and multi-level feature learning. Our proposed method, holistically-nested edge detection (HED), performs image-to-image prediction by means of a deep learning model that leverages fully convolutional neural networks and deeply-supervised nets. HED automatically learns rich hierarchical representations (guided by deep supervision on side responses) that are important in order to resolve the challenging ambiguity in edge and object boundary detection. We significantly advance the state-of-the-art on the BSD500 dataset (ODS F-score of .782*) and the NYU Depth dataset (ODS F-score of* .746*), and do so with an improved speed (*0.4 *second per image) that is orders of magnitude faster than some recent CNN-based edge detection algorithms.*

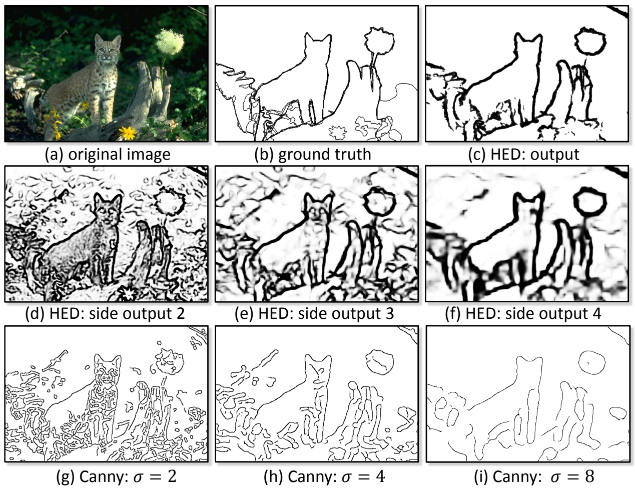

Figure 1. Illustration of the proposed HED algorithm. In the first row: (a) shows an example test image in the BSD500 dataset [26]; (b) shows its corresponding edges annotated by human subjects; (c) displays the result by HED. In the second row: (d), (e), and (f), respectively, show side edge responses from layers 2, 3, and 4 of our convolutional neural networks. In the third row: (g), (h), and (i), respectively, show edge responses from the Canny detector [4] at the scales $\sigma = 2.0$, $\sigma = 4.0$, and $\sigma = 8.0$. HED show its clear advantage in consistency over Canny.

1. Introduction

In this paper, we tackle the problem of detecting edges and object boundaries in natural images, which is both fundamental and of great importance to a variety of computer vision areas ranging from traditional tasks such as visual saliency, segmentation, object detection/recognition, tracking and motion analysis, medical imaging, structure-from-motion and 3D reconstruction, to modern applications like autonomous driving, mobile computing, and image-to-text analysis. It has been long understood that precisely localizing edges in natural images involves visual perception of various "levels" [16, 25]. A relatively comprehensive data collection and cognitive study [26] shows that while different subjects do have somewhat different preferences regarding where to place the edges and boundaries, there was nonetheless impressive consistency between subjects, e.g. reaching F-score 0.80 in the consistency study [26].

The history of computational edge detection is extremely rich; we now highlight a few representative works that have proven to be of great practical importance. Broadly speaking, one may categorize works into a few groups such as I: *early pioneering methods* like the Sobel detector [18], zero-crossing [25, 35], and the widely adopted Canny detector [4]; methods driven by II: *information theory* on top of features arrived at through careful manual design, such as Statistical Edges [20], Pb [26], and gPb [1]; and III: *heavily learning-based* methods that are still reliant on features of human design, such as BEL [5], Multi-scale [28], Sketch Tokens [22], and Structured Forests [6]. In addition, there has been a recent wave of development using *Convolutional Neural Networks* that emphasize the importance of automatic hierarchical feature learning, including N^4-Fields [9], DeepContour [32], DeepEdge [2], and CSCNN [17]. Prior to this explosive development in deep learning, the Structured Forests method (typically abbreviated

SE) [6] emerged as one of the most celebrated systems for edge detection, thanks to its state-of-the-art performance on the BSD500 dataset [26] (with, e.g., F-score of .746) and its practically significant speed of 2.5 frames per second. Recent CNN-based methods [9, 32, 2, 17] have demonstrated promising F-score performance improvements over SE. Even so, there remains room for improvement in these CNN-based methods, certainly in F-score performance, but even more so in speed — at present, time to make a prediction ranges from several seconds [9] to a few hours [2] (even when using modern GPUs).

Here, we develop an end-to-end edge detection system, holistically-nested edge detection (HED), that automatically learns the type of rich hierarchical features that are crucial if we are to approach the human ability to resolve ambiguity in natural image edge and object boundary detection. We acknowledge that it is slightly inaccurate to use the term "nested" in referring to those meaningful edge maps produced as side outputs — we intend to emphasize that many parts of the path along which each prediction is made are common to each of these edge maps, with successive edge maps including more. This integrated learning of hierarchical features is in distinction to previous multi-scale approaches [38, 39, 28] in which scale-space edge fields are neither automatically learned nor connected hierarchically. Figure 1 gives an illustration of an example image together with the human subject ground truth annotation, as well as results by the proposed HED edge detector (and also the responses in the individual layers), and results from the Canny edge detector with different scale parameters. Canny detection results at different scales are not only not directly connected, they also exhibit edge shift and inconsistency.

The proposed holistically-nested edge detection (HED) tackles two critical issues: (1) holistic image training and prediction; and (2) nested multi-scale feature learning, inspired by the fully convolutional neural networks [24] and by the deeply-supervised nets [21] that touch on, respectively, image-to-image classification and deep layer supervision (to "guide" early classification results). We find that the favorable characteristics of these underlying techniques manifest in HED being both accurate and computationally efficient.

2. Holistically-Nested Edge Detection

In this section, we describe in detail the network structure of our proposed edge detection system. We start by discussing related neural-network-based approaches, particularly those that emphasize multi-scale and multi-level feature learning. The task of edge and object boundary detection is intrinsically challenging. After decades of research, there have emerged a number of properties that a researcher in the field might generally agree are key and that are likely to play a role in a successful system: (1) carefully designed and/or learned features [26, 5], (2) multi-scale response fusion [38, 30, 28], (3) engagement of different levels of visual perception [16, 25, 37, 15] such as mid-level Gestalt law information [7], (4) structural information [6] and context [36], (5) holistic image prediction [23], (6) 3D geometry [13], and (7) occlusion boundaries [14].

The Structured Forests method [6] primarily focusses on three of these aspects: using a large number of manually designed features (property 1), fusing multi-scale responses (property 2), and incorporating structural information (property 4). A recent wave of work using CNNs for patch-based edge prediction [9, 32, 2, 17] share an alternative common thread that focusses on three aspects: automatic feature learning (property 1), multi-scale response fusion (property 2), and possible engagement of different levels of visual perception (property 3). However, due to the lack of deep supervision (that we include in our method), the multi-scale responses produced at the hidden layers in [2, 17] are less semantically meaningful, since feedback must be back-propagated through the intermediate layers. More importantly, their patch-to-pixel or patch-to-patch strategy results in significantly downgraded training and prediction efficiency. With some slight abuse of the term "holistically-nested", we emphasize here that we are producing an end-to-end edge detection system, a strategy inspired by fully convolutional neural networks [24], but with additional deep supervision on top of trimmed VGG nets [34] (shown in Figure 3). In the absence of deep supervision, a fully convolutional network [24] (FCN) produces a less satisfactory result (e.g. F-score .745 on BSD500) than HED, since edge detection demands highly accurate edge pixel localization. One thing worth mentioning is that our image-to-image training and prediction strategy still has not explicitly engaged contextual information, since constraints on the neighboring pixel labels are not directly enforced in HED. In addition to the speed gain over patch-based CNN edge detection methods, the performance gain is largely due to the following three aspects: (1) FCN-like image-to-image training allows us to simultaneously train on significantly larger amount of samples; (2) deep supervision in our model guides the learning of more transparent features; (3) upsampling in the side-output layers implicitly introduce guides the learning of more global information.

2.1. Existing multi-scale and multi-level NN

Due to the nature of hierarchical learning in the deep convolutional neural networks, the concept of multi-scale and multi-level learning might differ from situation to situation. For example, multi-scale learning can be "inside" the neural network, in the form of increasingly larger receptive fields and downsampled (strided) layers. In this "inside" case, the feature representations learned in each layer are naturally multi-scale. On the other hand, multi-scale

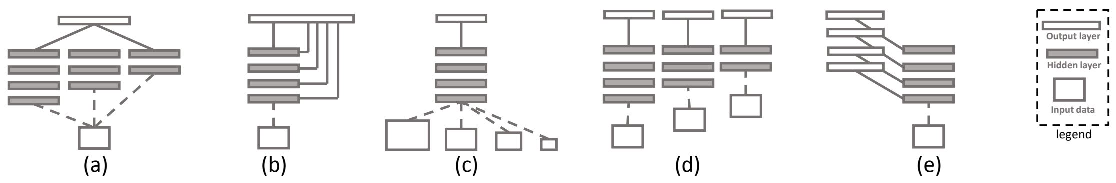

Figure 2. Illustration of different multi-scale deep learning settings. (a) is multi-stream architecture; (b) is skip-layer net architecture; (c) is a single model running on multi-scale inputs; (d) separate training of different networks; (e) is our proposed holistically-nested architectures, where multiple side outputs are added.

learning can be "outside" of the neural network, for example by "tweaking the scales" of input images. While these two variants have some notable similarities, we have seen both of them applied to different tasks.

We continue by next formalizing the possible configurations of multi-scale deep learning into 4 categories, namely, *multi-stream* learning, *skip-net* learning, a *single model* running on multiple inputs, and training of *independent* networks. Having these possibilities in mind will help make clearer the ways in which our proposed *holistically-nested* network approach differs from previous efforts and will help to highlight the important benefits in terms of representation and efficiency.

Multi-stream learning [3] [27] A typical multi-stream learning architecture is illustrated in Fig 1 (a). Note that the multiple (parallel) network streams have different numbers of parameters and receptive field sizes, and thus correspond to multiple scales. Input data are simultaneously fed into multiple streams, after which the concatenated feature responses produced by the various streams are fed into a global output layer to produce the final result.

Skip-layer network learning Examples of this form of network are [24][2][31] [9]. The key concept in "skip-layer" network learning is shown in Fig 1 (b). Instead of training multiple parallel streams, the topology for the skip-net architecture centers on a primary stream. Links are added to incorporate the feature responses from different levels of the primary network stream, and these responses are then combined in a shared output layer.

The common point in the two settings above is that, in both of the architectures, there is only one output layer so a single prediction is produced. However, in edge detection, it is often favorable (and, indeed, prevalent) to obtain multiple predictions and average the edge maps together.

Single model on multiple inputs To get multi-scale predictions, one can also run a single network on multiple (scaled) input images, illustrated in Fig 1 (c). This strategy can happen at both the training stage (as data augmentation) and at the testing stage (as "ensemble testing"). This approach is particularly common in non-deep-learning based methods [6]. Note that ensemble testing impairs the prediction efficiency of learning systems, with deeper models[2][9] possibly suffering more.

Training independent networks As an extreme variant to Fig 1 (a), one might pursue Fig 1 (d), in which multi-scale predictions are made by training multiple independent networks with different depths and different output loss layers. This is clearly impractical considering the factor by which this duplication would multiply the amount of resources required for training.

Holistically-nested networks We list all of these variants to help clarify the distinction between existing approaches and our proposed holistically-nested network approach, illustrated in Fig 1 (e). As one may already notice, there is significant redundancy in existing approaches, both as regards representational power and computational efficiency. Our proposed holistically-nested network is a relatively simple variant that is able to produce predictions from multiple scales. The architecture can be interpreted as a "holistically-nested" version of the "independent networks" approach in Fig 1 (d); this perspective is what motivates our name. Our architecture comprises a single-stream deep network with multiple side outputs. We note that this architecture resembles several previous works, particularly the deeply-supervised net approach in which the authors show that hidden layer supervision can improve both optimization and generalization for image classification tasks.

2.2. Formulations

Here we formulate our approach for edge prediction. We denote our input training data set by $S = \{(\mathbf{I}_i, \mathbf{G}_i), i = 1 \ldots N\}$, where sample $\mathbf{I}_i$ denotes the raw input image and $\mathbf{G}_i$ denotes the corresponding ground truth binary edge map for sample $\mathbf{I}_i$. We subsequently drop the subscript i for notational simplicity, since we consider each image holistically and independently. The goal of our edge detection framework is to learn feature layers that minimize the side output layer classification error. Suppose in the network we have M side-output layers. Each side-output layer is associated with a classifier, in which the corresponding weights are denoted as $\mathbf{w} = (\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(M)})$. For simplicity, we denote the collection of all the other network layer parameters as $\mathbf{W}$. We consider the objective function

$$\begin{aligned}\mathcal{L}_{\text{side}}(\mathbf{I}, \mathbf{G}, \mathbf{W}, \mathbf{w}) &= \sum_{m=1}^{M} \alpha_m \ell_{\text{side}}(\mathbf{I}, \mathbf{G}, \mathbf{W}, \mathbf{w}^{(m)}) \\ &= \sum_{m=1}^{M} \alpha_m \Delta(\hat{\mathbf{G}}^{(m)}, \mathbf{G}; \mathbf{W}, \mathbf{w}),\end{aligned} \tag{1}$$

where ℓ_{side} denotes the image-level loss function for side-outputs, $\hat{\mathbf{G}}^{(m)}$ is the (predicted) edge map produced by side-output layer m, upsampled to original size when necessary, Δ is an "energy function", (e.g. cross-entropy) computing the loss of the predicted edge map over the ground truth target, and α_m is a hyper-parameter controlling the loss weight for each individual side-output layer.

Under the hood of image-to-image training, the loss function is computed over all pixels in an image sample $\mathbf{I}$. For a typical natural image, the distribution of edge/non-edge pixels is heavily biased: 90% of the ground truth is non-edge. A cost-sensitive loss function is proposed in [17], where additional trade-off parameters are introduced for biased sampling. Here instead we use a simpler strategy to automatically balance the loss between positive/negative classes. We introduce a class-balancing weight β_j on a per-pixel term basis, where index j is over the image spatial dimensions of image $\mathbf{I}$. Then we use this class-balancing weight as a simple way to offset this imbalance between edge/non-edge. Specifically we define the following class-balanced cross-entropy loss function used in Equation (1)

$$\begin{aligned}\Delta = &- \beta_j \sum_{j=1}^{|\mathbf{I}|} \mathbf{G}_j \log \Pr(\mathbf{G}_j = 1 | , \mathbf{I}_j, \mathbf{W}, \mathbf{w}) \\ &- (1-\beta_j) \sum_{j=1}^{|\mathbf{I}|} (1-\mathbf{G}_j) \log \Pr(\mathbf{G}_j = 0 | , \mathbf{I}_j, \mathbf{W}, \mathbf{w}),\end{aligned} \tag{2}$$

where we denote $|\mathbf{I}|$,$|\mathbf{I}|_-$ and $|\mathbf{I}|_+$ as total number of all pixels, non-edge (negative) pixels and edge (positive) pixels in image $\mathbf{I}$, respectively. $\beta_j = \frac{|\mathbf{I}|_-}{|\mathbf{I}|}$ (when $\mathbf{I}_j$ is a positive) and $1 - \beta_j = \frac{|\mathbf{I}|_+}{|\mathbf{I}|}$ (when $\mathbf{I}_j$ is a negative).

2.3. Trimmed Network for Edge Detection

The choice of hierarchy for our framework deserves some thought. We need the architecture (1) to be deep, so as to efficiently generate perceptually multi-level features; and (2) to have multiple stages with different strides, so as to capture the intrinsic scales of edge maps. We must also keep in mind the potential difficulty in training such deep neural networks with multiple stages when starting from scratch.

Recently, VGGNet [34] achieved state-of-the-art performance in the ImageNet image classification challenge. We find that the VGGNet architecture design is a satisfactory starting point, since it is very deep (16 convolutional layers), dense (stride-1 convolutional kernels), and has multiple stages (five 2-strided downsampling layers). Other recent work [2] demonstrates that fine-tuning deep neural networks pre-trained on general image classification problem can be helpful to a low-level edge detection task. We find that the architecture designed in VGGNet is well-suited to our needs, although we do make the following modifications: (a) we connect our side output layer to the last convolutional layer in each stage, respectively conv1_1, conv2_2, conv3_3, conv4_3, conv5_3. The side-output layer is implemented as a convolutional layer with kernel size 1 and number of outputs 1. Therefore the receptive field size of each of these convolutional layers is identical to the corresponding side-output layer; (b) we cut the last stage of VGGNet, including all the fully connected layers and the 5th pooling layer. The reason for "trimming" the VGGNet is two-fold. Firstly, because we are expecting meaningful side outputs with different scales, a layer with stride 32 yields a too-small output plane with the consequence that the interpolated prediction map will be too fuzzy to utilize. Second, the fully connected layers (even when recast as convolutions) are computationally intensive, so that trimming pooling layer 5 and beyond can significantly reduce the memory/time cost during both training and testing. At the end of this process, our HED network architecture has 5 stages, with strides 1, 2, 4, 8 and 16, respectively, and with different receptive field sizes, all nested in the VGGNet. See Table 1 for a summary of configurations of receptive field and neuron stride.

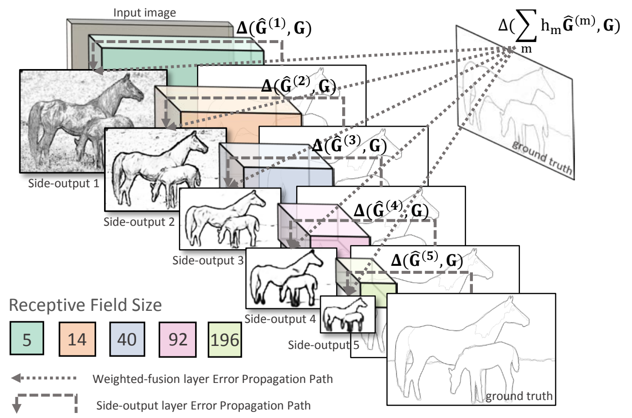
Input image
$\Delta(\hat{\mathbf{G}}^{(1)}, \mathbf{G})$
$\Delta(\sum_m h_m \hat{\mathbf{G}}^{(m)}, \mathbf{G})$
$\Delta(\hat{\mathbf{G}}^{(2)}, \mathbf{G})$
$\Delta(\hat{\mathbf{G}}^{(3)}, \mathbf{G})$
$\Delta(\hat{\mathbf{G}}^{(4)}, \mathbf{G})$
$\Delta(\hat{\mathbf{G}}^{(5)}, \mathbf{G})$
ground truth
Side-output 1
Side-output 2
Side-output 3
Side-output 4
Side-output 5
ground truth
Receptive Field Size
5
14
40
92
196
Weighted-fusion layer Error Propagation Path
Side-output layer Error Propagation Path

Figure 3. Illustration of the our network architecture for edge detection, highlighting the error backpropagation paths. Side-output layers are inserted after convolutional layers. Deep supervision is imposed at each side-output layer, guiding the side-outputs towards edge predictions with the characteristics we desire. The outputs of HED are multi-scale and multi-level, as the side-output plane size becomes smaller, and the receptive field size becomes larger. In the setting of edge detection, one weighted-fusion layer can be added to automatically learn how to combine outputs from multiple scales. The whole network is trained with multiple error propagation paths (dashed lines).

2.4. Weighted-fusion Layer

One natural question is how to best utilize the side-outputs produced by our framework; while the most straightforward approach is to simply average all the edge map predictions, one can also learn a classifier to optimize the combination weights, either at pixel level (locally connected) or at image level. We add a "weighted-fusion" layer

Table 1. The receptive field and stride size in VGGNet [34]. The bolded convolutional layers are linked to additional side-output layers.

layer	**c1_2**	p1	**c2_2**	p2	**c3_3**
rf size	**5**	6	**14**	16	**40**
stride	**1**	2	**2**	4	**4**
layer	p3	**c4_3**	p4	**c5_3**	p5
rf size	44	**92**	100	**196**	212
stride	8	**8**	16	**16**	32

to the network and link all the side-output layer predictions and (simultaneously) learns the fusion weight during training. Denoting this fusion weight as $\mathbf{h} \in \mathbb{R}^{\mathbf{M}}$, our loss function at the fusion layer $\mathcal{L}_{\text{fuse}}$ becomes

$$\mathcal{L}_{\text{fuse}}(\mathbf{I}, \mathbf{G}, \mathbf{W}, \mathbf{w}) = \Delta(\sum_{m=1}^{M} h_m \hat{\mathbf{G}}^{(m)}, \mathbf{G}; \mathbf{W}, \mathbf{w})$$

With Δ still being the class-balancing cross-entropy loss function as defined in Equation (2). The overall loss function then becomes

$$\mathcal{L}(\mathbf{I}, \mathbf{G}, \mathbf{W}, \mathbf{w}) = \mathcal{L}_{\text{side}}(\mathbf{I}, \mathbf{G}, \mathbf{W}, \mathbf{w}) + \mathcal{L}_{\text{fuse}}(\mathbf{I}, \mathbf{G}, \mathbf{W}, \mathbf{w})$$

We train this loss via standard back-propagation stochastic gradient descent. See section 3 for detailed hyper-parameter and experiment settings.

Hidden-layer supervision Since we incorporate a weighted-fusion output layer that connects each side-output layer, one might argue that the hidden supervision terms (specifically, $\ell_{\text{side}}(\mathbf{I}, \mathbf{G}, \mathbf{W}, \mathbf{w})$) are unnecessary in this formulation, since now the whole network is path-connected and the output-layer parameters can be updated by back-propagation through the weighted-fusion layer error propagation path. In contrast to this view, we argue that deep supervision is important to obtain desired edge maps. The key characteristic of our proposed network is that each network layer is supposed to play a role as a singleton network responsible for producing an edge map at a certain scale. We show qualitative results based on the two variants discussed above: training with both weighted-fusion supervision and deep supervision, or training with weighted-fusion supervision only. We observe that with deep supervision, the nested side-outputs are meaningful and agree with expectations, insofar as the successive edge map predictions are progressively coarse-to-fine, local-to-global. On the other hand, training with only the weighted-fusion output loss gives edge predictions that lack any such discernible order. We also find that many critical edges are lost at the high-level side output. Our claim is further supported quantitatively by the experimental results on benchmark dataset, which will be discussed later.

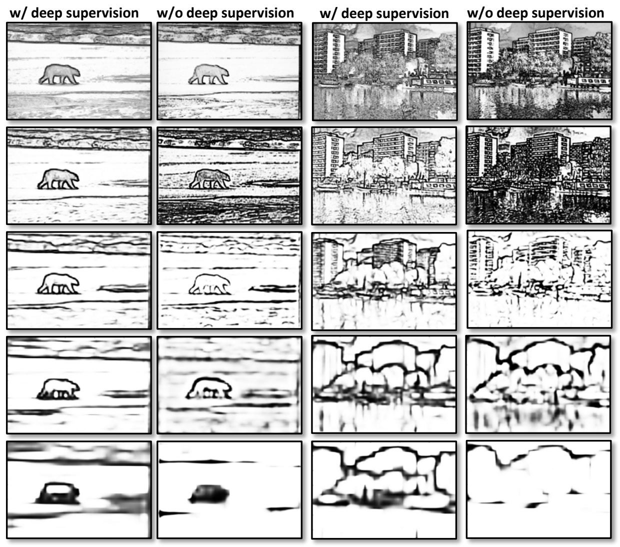

Figure 4. Two examples illustrating how deep supervision helps side-output layers produce multi-scale dense predictions. Note that in the left column, the side outputs become progressively coarser and more “global”, while critical object boundaries are preserved. In the right column, the predictions tends to lack any discernible order (e.g. in layers 1 and 2), and many boundaries are lost in later stages.

3. Experimental Framework

Model Parameters In contrast to fine-tuning a classification net to perform a high-level computer vision task like fine-grained image classification or semantic segmentation, adapting the classification net to a low-level edge detection task is much more challenging. Differences in data distribution, ground truth distribution, and loss function all contribute to difficulties in network convergence, even with the initialization of the pre-trained model. We first use a validation set and follow the evaluation strategy used in [6] to tune the deep model hyper-parameters. The parameters we tune (and the values we choose) are: the size of the mini-batch (10), learning rate (1e-6), the loss-weight α_m for each side-output layer (1), momentum (0.9), initialization of the nested filters (0), initialization of fusion layer weight (1/5), weight decay (0.0002), training iterations (10,000; reduce learning rate by 1/10 after 5,000). We focus on the convergence behavior of the network, since we observe that whenever training converges, the deviations in F-score on the validation set tend to be very small. In order to investigate whether including additional nonlinearity helps, we also consider a setting in which we add an additional layer (with 50 filters and a ReLU) before each side-output layer; we find that performance here is worse. On another note, we observe that our nested multi-scale framework is insensitive to input image scales; during our training process, we take advantage of this by resizing all the images to 400×400 to reduce GPU memory usage and to take advantage of efficient batch processing. In later experiments, we fix the values of all parameters discussed above and explore other variants of HED on the fully-independent test set.

Consensus Sampling In our approach, we duplicate the ground truth at each side-output layer and resize the (down-sampled) side output to its original scale. Thus there exists a mismatch in the high-level side-outputs: The edge predictions are coarse and global, while the ground truth still contains many weak edges that could even be considered as noise. This issue leads to problematic convergence behavior, even with the help of a pre-trained model. We observe that this mismatch leads to back-propagated gradients that explode at the high-level side-output layers. We can, however, adjust how we make use of the ground truth labels in the BSDS dataset to combat this issue. Specifically, the ground truth labels are provided by multiple annotators and thus, implicitly, greater labeler consensus indicates stronger ground truth edges. We adopt a relatively brute-force solution: we assign a pixel a positive label if and only if it is labeled as positive by at least three annotators, and put all other positively labeled pixels with only one or two labeler consensus into the negative set. This addresses the problem of gradient explosion in high level side-output layers. For low level layers, reducing positive pixels brings additional robustness and prevents the network from being distracted by weak edges. Though not fully explored in our paper, we believe that careful handling of consensus levels of ground truth edge maps can lead to further improvement.

Data Augmentation Data augmentation has proven to be a crucial technique in deep networks. We rotate the images to 16 different angles and crop the largest rectangle in the rotated image; we also flip the image at each angle, thus augmenting the training set by a factor of 32. We find that augmenting the data to different scales is unnecessary. During testing we operate on an image in its original scale. We also note that "ensemble testing" (testing on rotated/flipped images and averaging the prediction) yields no improvements in either F-score or average precision.

3.1. Architecture alternatives

FCN and skip-layer architecture The topology used in the FCN model differs from our HED model in several aspects. As we have discussed, while FCN reinterprets the classification nets for pixel-wise prediction, it has only one single stream output. In FCN, the skip net structure is a DAG that combines coarse, high layer information with fine low layer information, without explicitly producing multi-scale output predictions. We explore how this architecture can be used for the edge detection task, under exactly the same experimental setup as our HED model. We first try to directly apply the FCN-8s model by replacing the loss function with cross-entropy for edge detection. This results in a model that achieves ODS=.725, OIS=.743 and AP = .680. This unsatisfactory result can be expected since this architecture is still not fine enough. We further explore whether the performance can be improved by adding even more links from low-level layers. We then create an FCN-2s network that adds additional links from the pool1 and pool2 layers, the performance improves to ODS=.745, OIS=.765, AP= .730. Still, directly applying FCN skip-net topology falls behind our proposed holistically nested architecture in edge detection tasks. We feel that, with heavy tweaking of the FCN, one might also be able to achieve competitive performance on edge detection, but we value the multi-scale side-outputs for their ability to provide additional flexibility, especially for edge detection.

Different pooling function Previous work [2] suggests that different pooling functions can have a major impact on the performance for edge detection. We conduct a controlled experiment in which all pooling layers are replaced by average pooling. In contrast to the observation in [2], we find using average pooling decrease the performance to ODS=.741.

In-network linear interpolation Built on top of [24], side-output prediction upsampling is implemented with in-network deconvolutional layers. We fix all the deconvolutional layers to linear interpolation. Although [24] points out that one can learn arbitrary interpolation functions, we find that learned deconvolutions provide no noticeable improvements in our experiments.

Training without deep supervision As we have discussed in previous section, the deep supervision plays an important role in HED. We test the model trained with/without deep supervision imposed. The model trained without deep supervision consistently performs worse (At least .01 in F-score and .02 in AP).

3.2. Implementation

We implement our framework in publicly available *Caffe* Library and build on top of the publicly available implementations of FCN[24] and DSN[21], thus relatively little engineering hacking is required. In our HED system the whole network is fine-tuned from an initialization with the VGG-16 Net pre-trained model.

Running time Training takes about 7 hours on a single NVIDIA K40 GPU. Testing is also efficient. For an 320×480 image, HED takes 400 ms to make an edge prediction, including the interface overhead. Many edge detectors improve performance by sacrificing efficiency (for example, by testing on input images from multiple scales and averaging the results).

4. Results

In this section we report the performance of our proposed algorithm.

BSDS500 dataset We perform the majority of the experiments on the Berkeley Segmentation Dataset and Bench-

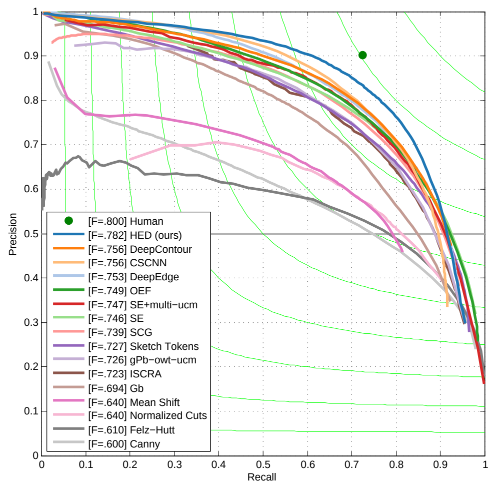

Figure 5. Results on BSDS500 dataset. Our proposed HED framework achieve the best result (ODS=.782). Compared to several recent CNN-based edge detectors, our approach is also orders of magnitude faster. See Table 2 for detailed information.

Table 2. Results on BSDS500. ∗BSDS300 results,†GPU time

	ODS	OIS	AP	FPS
Human	.80	.80	-	-
Canny	.600	.640	.580	15
Felz-Hutt [8]	.610	.640	.560	10
BEL [5]	.660∗	-	-	1/10
gPb-owt-ucm [1]	.726	.757	.696	1/240
Sketch Tokens [22]	.727	.746	.780	1
SCG [29]	.739	.758	.773	1/280
SE-Var [6]	.746	.767	.803	2.5
OEF [12]	.749	.772	.817	-
DeepNets [19]	.738	.759	.758	1/5†
N4-Fields [9]	.753	.769	.784	1/6†
DeepEdge [2]	.753	.772	.807	$1/10^3$†
CSCNN [17]	.756	.775	.798	-
DeepContour [32]	.756	.773	.797	1/30†
HED (ours)	**.782**	**.804**	**.833**	2.5†, 1/12

mark (BSDS 500) [1]. The dataset contains 200 training, 100 validation and 200 testing images. Each image has hand annotated ground truth contours. Edge detection accuracy is evaluated using three standard measures: fixed contour threshold (ODS), per-image best threshold (OIS), and the average precision (AP). Prior to evaluation, we apply a standard non-maximal suppression technique to our edge maps to obtain thinned edges. The results are shown in Figure 5 and Table 2.

Late merging to boost average precision We find that the weighted-fusion layer output gives best performance in F-score. However the average precision degrades compared to directly averaging all the side outputs. This is due to the fact that during training we are focusing on "global" object boundaries for the fusion-layer weight learning. However,

Table 3. Results of single and averaged side output in HED on the BSDS 500 dataset. Each individual side output contributes to the fused/averaged result. Note that the learned weighted-fusion (*Fusion-output*) achieves best F-score, while directly averaging all of the five layers (*Average 1-5*) produces better average precision. Merging those two readily available outputs further boost the performance.

	ODS	OIS	AP
Side-output 1	.595	.620	.582
Side-output 2	.697	.715	.673
Side-output 3	.738	.756	.717
Side-output 4	.740	.759	.672
Side-output 5	.606	.611	.429
Fusion-output	**.782**	**.802**	.787
Average 1-4	.760	.784	.800
Average 1-5	.774	.797	**.822**
Average 2-4	.766	.788	.798
Average 2-5	.777	.800	.814
Merged result	**.782**	**.804**	**.833**

because in HED all the side outputs are readily available after a single test, we can merge the fusion layer output with the directly averaged counterpart, at no extra cost, to compensate for the loss in average precision. This simple heuristic gives us the best performance across all measures that we report in Figure 5 and Table 2.

More training data Deep models drive advances in computer vision due to large learning capacity and huge training data. Unfortunately, in edge detection, we are limited by the number of training images available in the dataset. Here we want to explore whether adding more training data will allow us to further improve the results. To do this, we expand the training set by random sampling 100 images from the testing set. We then evaluate the result on the rest 100 test images. We report the averaged result over 5-fold trials.

From the result we observe that by only adding 100 training images, the performance improves from ODS=.782 to ODS=.797 (±.003), and nearly touches the human benchmark. We believe that, with even larger annotated dataset, the performance can be further improved.

Results discussion Table 2 and Figure 5 lists the precision-recall results obtained by HED relative to other competing methods. Table 3 summarizes the results produced by each individual side-output at different scales, as well as different combinations of the multi-scale edge maps. We want to emphasize that all the side-output predictions are obtained all in one pass, which enables us to fully investigate different configurations of combining the outputs at no extra cost. There are several interesting observations from the results. For instance, combination of predictions from multiple scales yields better performance, and all the side-output layers contribute to the result, either in F-score or averaged precision. To see this, in Table 3, the side-output layer 1 and layer 5 (the lowest and highest layers) achieve similar relatively low performance. One might expect these two side-output layers to not be useful in the averaged results. However this turns out not to be the case — for example, the Average 1-4 achieves ODS=.760 and incorporating the side-output layer 5, the averaged prediction achieves an ODS=.774. We find similar phenomenon when considering other ranges. As mentioned above, the predictions obtained using different combination strategies are complementary, and a late merging of the averaged predictions with learned fusion-layer predictions leads to the best result.

NYUDv2 Dataset The NYU Depth (NYUD) dataset [33] has 1449 RGB-D images. The dataset was used for edge detection in [29] and [10]. Here we follow the [6] and perform the experiment on the data set processed by [10]. the NYUD dataset is split into 381 training, 414 validation, and 654 testing images. All images are cropped to the same dimension, thus we train our network on full resolution images. Finally following the experimental setup in [11] and [6], the maximum tolerance allowed for correct matches of edge predictions to ground truth during evaluation increases from .0075 to .011.

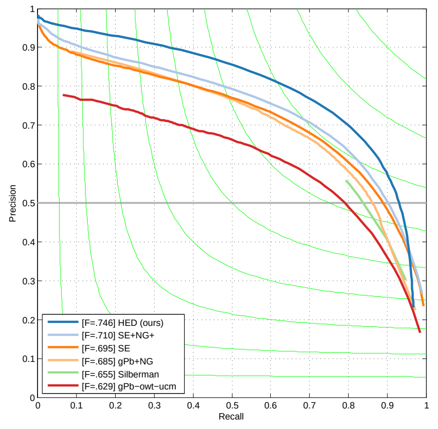

Figure 6. Precision/recall curves on NYUD dataset. Holistically-Edge Detection (HED) trained with RGB and HHA feature achieves the best result (ODS=.746). See Table 4 for additional information.

Table 4. Results on the NYUD dataset [33] †GPU time

	ODS	OIS	AP	FPS
gPb-ucm	.632	.661	.562	1/360
Silberman [33]	.658	.661	-	1/360+
gPb+NG[10]	.687	.716	.629	1/375
SE[6]	.685	.699	.679	5
SE+NG+[11]	.710	.723	.738	1/15
HED-RGB	.720	.734	.734	2.5†
HED-D	.682	.695	.702	2.5†
HED-RGBD	**.746**	**.761**	**.786**	1†

Depth information encoding Following the success in [11] and [24], we leverage the depth information by utilizing the HHA feature, which the depth information is embedded into three-channel feature, i.e. horizontal disparity, height above ground, and the angle of the local surface normal with the inferred gravity direction. We use the same architectures and parameter settings as the experiments on BSDS 500. We train two different models in parallel, on RGB images and HHA feature images, and report the results respectively. Then we directly average the RGB and HHA predictions as our final result leveraging RGB-D information. We have tried other approaches to incorporate the depth information, for example, train on the raw depth channel, or concatenate the depth channel with the RGB channel before the first convolutional layer. None of these attempts yields competitive performance compared to using HHA. The effectiveness of HHA feature shows that, although deep neural networks are capable of automatic feature learning, for depth data, carefully hand-designed features are still necessary, especially when very limited training data is available

Results discussion Table 4 and Figure 6 show the precision-recall evaluations of HED vs other competing methods. All the network structures for training are kept the same as for BSDS. During testing we use *Average2-4* prediction instead of Fusion-layer output as it yields the best performance. We do not perform the late merging, since by combining two sources of edge map predictions (RGB and HHA), the average precision is already high. Note that the results achieved by only using RGB modality have already bypassed all the previous approaches.

5. Discussion and Conclusion

In this paper, we have developed a new convolutional-neural-network-based edge detection system that demonstrates state-of-the-art performance on natural images at a speed of practical importance (e.g. 0.4 second using GPU and 12 seconds on CPU). Our algorithm builds on top of the ideas of fully convolutional neural networks and deeply-supervised nets. We also initialize our network structure and parameters by adopting a pre-trained trimmed VGGNets. Our method shows promising results in performing image-to-image learning and testing for edge detection by combining multi-scale and multi-level visual responses, even though explicit contextual and high-level information has not been enforced.

6. Acknowledgment

This work is supported by NSF IIS-1216528 (IIS-1360566) and NSF award IIS-0844566 (IIS-1360568). We gratefully thank Patrick Gallagher for helping improve this manuscript. We are grateful for the generous donation of the GPUs by NVIDIA.

References

[1] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *PAMI*, 33(5):898–916, 2011. 1, 7

[2] G. Bertasius, J. Shi, and L. Torresani. Deepedge: A multi-scale bifurcated deep network for top-down contour detection. In *CVPR*, 2015. 1, 2, 3, 4, 6, 7

[3] P. Buyssens, A. Elmoataz, and O. Lézoray. Multiscale convolutional neural networks for vision–based classification of cells. In *ACCV*. 2013. 3

[4] J. Canny. A computational approach to edge detection. *PAMI*, (6):679–698, 1986. 1

[5] P. Dollar, Z. Tu, and S. Belongie. Supervised learning of edges and object boundaries. In *CVPR*, volume 2, pages 1964–1971. IEEE, 2006. 1, 2, 7

[6] P. Dollár and C. L. Zitnick. Fast edge detection using structured forests. *PAMI*, 2015. 1, 2, 3, 5, 7, 8

[7] J. H. Elder and R. M. Goldberg. Ecological statistics of gestalt laws for the perceptual organization of contours. *Journal of Vision*, 2(4):5, 2002. 2

[8] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *IJCV*, 59(2):167–181, 2004. 7

[9] Y. Ganin and V. Lempitsky. N4-fields: Neural network nearest neighbor fields for image transforms. *arXiv preprint arXiv:1406.6558*, 2014. 1, 2, 3, 7

[10] S. Gupta, P. Arbelaez, and J. Malik. Perceptual organization and recognition of indoor scenes from rgb-d images. In *CVPR*, 2013. 8

[11] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik. Learning rich features from rgb-d images for object detection and segmentation. In *ECCV*, 2014. 8

[12] S. Hallman and C. C. Fowlkes. Oriented edge forests for boundary detection. *arXiv preprint arXiv:1412.4181*, 2014. 7

[13] D. Hoiem, A. A. Efros, and M. Hebert. Putting objects in perspective. *IJCV*, 80(1):3–15, 2008. 2

[14] D. Hoiem, A. N. Stein, A. A. Efros, and M. Hebert. Recovering occlusion boundaries from a single image. In *ICCV*, 2007. 2

[15] X. Hou, A. Yuille, and C. Koch. Boundary detection benchmarking: Beyond f-measures. In *CVPR*, 2013. 2

[16] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962. 1, 2

[17] J.-J. Hwang and T.-L. Liu. Pixel-wise deep learning for contour detection. *ICLR*, 2015. 1, 2, 4, 7

[18] J. Kittler. On the accuracy of the sobel edge detector. *Image and Vision Computing*, 1(1):37–42, 1983. 1

[19] J. J. Kivinen, C. K. Williams, N. Heess, and D. Technologies. Visual boundary prediction: A deep neural prediction network and quality dissection. In *AISTATS*, 2014. 7

[20] S. Konishi, A. L. Yuille, J. M. Coughlan, and S. C. Zhu. Statistical edge detection: Learning and evaluating edge cues. *PAMI*, 25(1):57–74, 2003. 1

[21] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. In *AISTATS*, 2015. 2, 6

[22] J. J. Lim, C. L. Zitnick, and P. Dollár. Sketch tokens: A learned mid-level representation for contour and object detection. In *CVPR*, 2013. 1, 7

[23] C. Liu, J. Yuen, and A. Torralba. Nonparametric scene parsing via label transfer. *PAMI*, 33(12):2368–2382, 2011. 2

[24] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *CVPR*, 2015. 2, 3, 6, 8

[25] D. Marr and E. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 207(1167):187–217, 1980. 1, 2

[26] D. R. Martin, C. 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

[27] N. Neverova, C. Wolf, G. W. Taylor, and F. Nebout. Multi-scale deep learning for gesture detection and localization. In *ECCV Workshops*, 2014. 3

[28] X. Ren. Multi-scale improves boundary detection in natural images. In *ECCV*. 2008. 1, 2

[29] X. Ren and L. Bo. Discriminatively trained sparse code gradients for contour detection. In *NIPS*, pages 584–592, 2012. 7, 8

[30] D. L. Ruderman and W. Bialek. Statistics of natural images: Scaling in the woods. *Physical review letters*, 73(6):814, 1994. 2

[31] P. Sermanet, S. Chintala, and Y. LeCun. Convolutional neural networks applied to house numbers digit classification. In *ICPR*, 2012. 3

[32] W. Shen, X. Wang, Y. Wang, X. Bai, and Z. Zhang. Deep-contour: A deep convolutional feature learned by positive-sharing loss for contour detection draft version. In *CVPR*, 2015. 1, 2, 7

[33] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*. 2012. 8

[34] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 2, 4, 5

[35] V. Torre and T. A. Poggio. On edge detection. *PAMI*, (2):147–163, 1986. 1

[36] Z. Tu. Auto-context and its application to high-level vision tasks. In *CVPR*, 2008. 2

[37] D. C. Van Essen and J. L. Gallant. Neural mechanisms of form and motion processing in the primate visual system. *Neuron*, 13(1):1–10, 1994. 2

[38] A. P. Witkin. Scale-space filtering: A new approach to multi-scale description. In *ICASSP*, volume 9, pages 150–153, 1984. 2

[39] A. L. Yuille and T. A. Poggio. Scaling theorems for zero crossings. *PAMI*, (1):15–25, 1986. 2