arXiv:2005.13862v1 [cs.CV] 28 May 2020

TRADITIONAL METHOD INSPIRED DEEP NEURAL NETWORK FOR EDGE DETECTION

Jan Kristanto Wibisono and Hsueh-Ming Hang

Department of Electronics Engineering, National Chiao Tung University,Taiwan
jankristanto.03g@g2.nctu.edu.tw, hmhang@nctu.edu.tw

ABSTRACT

Recently, Deep-Neural-Network (DNN) based edge prediction is progressing fast. Although the DNN based schemes outperform the traditional edge detectors, they have much higher computational complexity. It could be that the DNN based edge detectors often adopt the neural net structures designed for high-level computer vision tasks, such as image segmentation and object recognition. Edge detection is a rather local and simple job, the over-complicated architecture and massive parameters may be unnecessary. Therefore, we propose a traditional method inspired framework to produce good edges with minimal complexity. We simplify the network architecture to include Feature Extractor, Enrichment, and Summarizer, which roughly correspond to gradient, low pass filter, and pixel connection in the traditional edge detection schemes. The proposed structure can effectively reduce the complexity and retain the edge prediction quality. Our TIN2 (Traditional Inspired Network) model has an accuracy higher than the recent BDCN2 (Bi-Directional Cascade Network) but with a smaller model.

***Index Terms*—** Edge detection, traditional edge detector, deep neural net, CNN

1. INTRODUCTION

Edge detection is trying to find the high-contrast boundary that separates two regions. Edge detector is a fundamental operator in image processing. Along with the development of deep learning (DL), the development of Deep Neural Network (DNN) based edge detection also shows the advantage of high accuracy performance. On the BSDS500 dataset [1], the traditional methods typically only achieve 0.59 ODS F-measure, but the DL-based methods can achieve 0.828 ODS [2]. The current trend is to develop lighter edge detectors that maintain the detection accuracy. Fig. 1 shows both the detection accuracy and complexity (model size) of several well-known DNN based methods. Since the first deep edge detection was proposed, the ODS accuracy has increased from 0.782 (HED [3]) to 0.828 (BDCN [2]). Often, the DNN based edge detectors adopt the neural net structures designed for high-level computer vision tasks, such as image segmentation and object recognition. Therefore, they contain millions of parameters. Edge detection is a rather local and simple job; the over-complicated architecture and massive parameters may be unnecessary. For example, many researchers use VGGNet (Visual Geometry Group) [4] as their feature-based extractor, which was originally designed for image classification problem. Consequently, their models have the capacity to extract a large number of features needed for image classification, but this large capacity could be overkill for edge detection purpose. In our opinion, edge

The source code is available at https://github.com/janctu/TIN

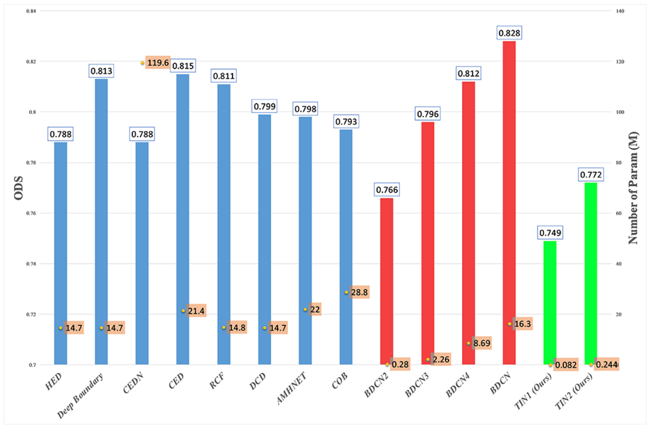

Fig. 1. Comparison of complexity and accuracy performance among various edge detection schemes. Our proposed methods (Green). BDCN family (Red). Other methods (Blue). ODS (Transparent label). Number of Parameter (Orange label).

detection is a much simpler job compared to image classification or semantic segmentation. In semantic segmentation, feature extractor is required to be able to recognize many different patterns of objects, whereas the edge detection only recognizes edge or non-edge. The traditional edge detectors [5, 6] typically only use gradients together with low-pass filtering and pixel connection. This paper will focus on how to design an effective and lightweight deep learning framework to detect edges. We aim to reduce the complexity and retain the edge detection accuracy at the same time. In summary, in this paper, we propose a set of DNN based edge detectors which are inspired by the traditional methods. Our systems contain three basic modules: Feature Extractor, Enrichment, and Summarizer, which roughly correspond to gradient, low pass filter, and pixel connection in the traditional edge detection schemes. We compare the proposed method with the state-of-the-art methods. Our method has a much lower complexity at about the same edge detection accuracy.

2. RELATED WORK

In the last few decades, many Edge Detection approaches have been proposed. We will highlight a few significant works. Sobel [5] and Canny [6] are the early works of Edge Detection. Sobel detection is calculating the magnitude of the gradient in an image using a set of 3x3 filters. The next generation is the learning-based methods [7, 8, 9, 10, 11, 12, 13]. The learning-based methods often extract low-level features and use trained classifiers based on features to produce object boundaries. Even though these learning-based methods produce a much better performance compared to the conventional gradient methods, often their performance depends on how well the hand-crafted features are designed. Recently, edge detection has been developed based on the deep convolutional neural networks. DeepEdge [14] and DeepCountour [15] are early learning-based edge detection schemes. Even though they were built on top of the deep neural network, they still adapted the notion of patches from the structured forest [12] and sketch tokens [11]. Different from the patches based on deep learning, HED [3] is an end-to-end fully convolutional neural network that accepts images as input and outputs the edge probability for every pixel. HED uses VGGNet [4] for the feature extraction. It fuses the all side-outputs of VGGNet features and minimizes the weighted cross-entropy loss function. HED is one of the most influential papers in the DNN-based edge detection. Then, Crisp Edge Detector (CED) [16] and Richer Convolutional Features (RCF) [17] were designed to enrich the use of the side-outputs. Also, there is an LPCB (Learning to Predict Crips Boundary) scheme [18], which adjusts edge prediction by introducing a fusion loss function between the Dice coefficient [19] and the cross-entropy loss. The most recent work we found is BDCN [2], which generates multiple ground-truth maps using different side-outputs. It is a family of schemes of different complexities and accuracies. It outperforms the other schemes at about the same complexities.

3. PROPOSED METHOD

3.1. Motivation

A traditional edge detector often consists of gradient calculation, low-pass filter, pixel connection, and non-maximum suppression (NMS). Inspired by the traditional structure in building our system, our system consists of Feature Extractor, Enrichment, and Summarizer.

3.2. Feature Extractor

We start from a simple feature extractor, which is a 3x3 convolutional neural network layer. We mimic the gradient operators and after a few experiments, we settle on 16 feature channels and they are initialized with the 16 directional gradient kernels. The other CNN layers are initialized with zero-mean, 0.01 standard deviation Gaussian and zero biases. Although the feature extractor was designed to simulate the gradient operators, after training, some output feature maps show edges, while the other ones show, kind of, segmentation maps. Fig. 2 shows the 16 output maps from the first feature extractor. (Point A on Fig. 6)

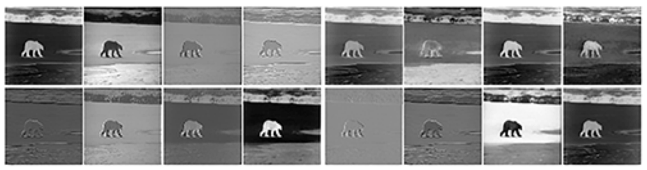
Fig. 2. 16 feature maps from the 1st Feature Extractor

3.3. Enrichment

In a traditional edge detector, low-pass filters and the other more complicated schemes are often used to remove the noise or tiny/isolated edge candidates. We use dilated convolution[20] to build multiple scale filtering. The dilated convolution can capture a larger receptive field with fewer parameters. We call this module Enrichment because it also extracts object information in addition to edges. Fig. 3 shows the design of our Enrichment module. The 3x3 means the filter size is 3x3. The 16 means the number of input and the 32 means the number of output / number of filter. The outputs of each dilated filter are added together at the end. Fig. 4 shows the 32 channel outputs of the Enrichment module (Point B on Fig. 6). The outputs are generally more blurred than their inputs (Feature Extractor outputs, Fig. 2). Some outputs contain only some portions of object edges; they seem to be complimentary to each other.

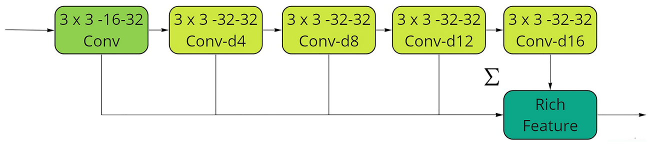

Fig. 3. Enrichment component

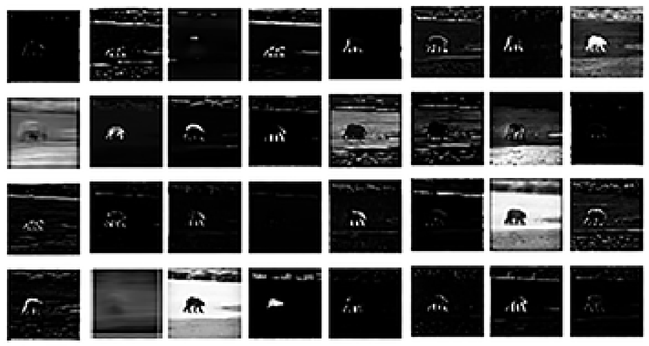
Fig. 4. Intermediate outputs from Enrichment

3.4. Summarizer

The last module tries to summarize the features generated by the Enrichment modules and to produce the final edges. We use eight 1x1 convolutional layers together with a sigmoid activation function. Fig. 5(a) shows the 8 channel outputs of Summarizer 1 (Point C on Fig. 6), and (b) shows the final output of two summarizers (Point D on Fig. 6).

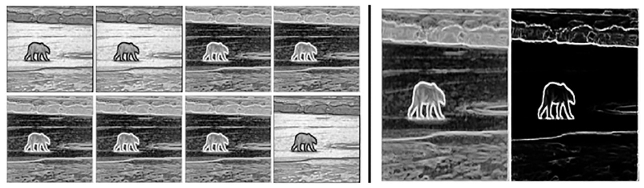
Fig. 5. (a) Outputs of Summarizer 1 (b) Final Output

3.5. Loss Function

Loss Function plays a critical role in our system. Because the edge pixels are much fewer than the non-edge pixels, we use a weighted cross-entropy loss [3]. We compute the loss of each pixel prediction with respect to the ground-truth as follows.

$$L(X_i;W) = \begin{cases} \alpha.log(1-P(X_i;W)), & \text{if } y_i = 0 \\ 0, & \text{if } 0 < y_i < th \\ \beta.log(P(X_i;W)), & \text{otherwise} \end{cases} \quad (1)$$

where y_i, X_i,W are the ground-truth, input, and weights, respectively. $P(X)$ is the sigmoid function, and th is a threshold of value 64 in our case. By set the th, we ignore some weak edges because it is not significant and may confuse the network.

$$\alpha = \gamma.\frac{Y_+}{Y} \qquad \beta = \frac{Y_-}{Y} \quad (2)$$

Y_+ and Y_- denote the numbers of edge and non-edge ground truth pixels, respectively. The parameter γ is used to balance between edges and non-edges. Here, we use $y = 1.1$ to emphasis more on edge. This loss function is applied to all the side-outputs and the fused outputs. Thus the total loss function is Eq. 3, where K is the number of stages.

$$LL(W) = \sum_{k=1}^{K}\sum_{i=1}^{I} L(X_i^k;W) + \sum_{i=1}^{I} L(X_i^{fuse};W) \quad (3)$$

3.6. Post-processing

Similar to [2, 3, 16, 17, 18], we use multi-scale testing and non-maximum suppression (NMS) as our post-processing. First, we resize the input image to three different resolutions, 0.5x,1x, and 1.5x, then feed them into the network. We resize the outputs into the original size and average them to produce the final result. The purpose of NMS is to thin the edge prediction maps.

3.7. Traditional Method Inspired Edge Detectors

We propose two architectures using the components discussed earlier. We called them Traditional Inspired Network 1 (TIN 1) and Traditional Inspired Network 2 (TIN 2). TIN2 is simply a stack of two TIN1, but the stage one feature map is downsampled (max-pooling) in half before entering the second stage. Fig. 6 shows the architecture of TIN1. Fig. 7 is TIN2. Our input is the original color image without any preprocessing. The input is passed to the Feature Extractor 1, which is 16 filters of 3x3 convolution as we described before. These features go to the Feature Extractor 2 with random initialization. Each Feature Extractor passes its outputs to the Enrichment and Summarizer modules. At the end, we add the Summarizer output features and fuse them by a 1x1 convolution. In training, the

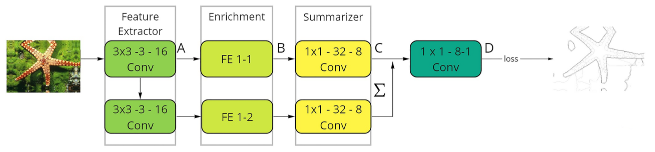

Fig. 6. TIN1 Architecture

loss function is optimized to tune the parameters. The TIN2 (Fig.7) Feature Extractor has 64 filters. TIN2 produces multiple prediction outputs. To fuse the outputs of both stages, we apply the bilinear interpolation to the second stage outputs. Concatenating stage 1 outputs and the interpolated stage 2 outputs, we form the final edge map using another 1x1 convolution. The bilinear interpolation may be replaced by a deconvolutional layer but it produces similar results in our experiment.

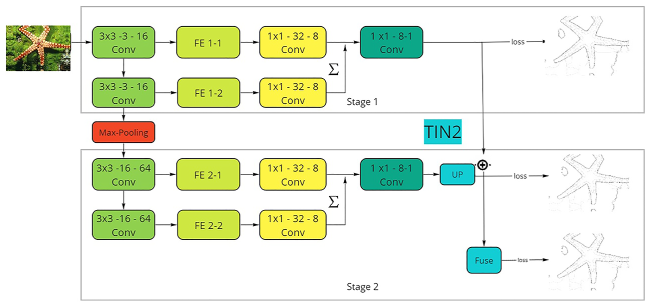

Fig. 7. TIN2 Architecture

4. EXPERIMENTS

4.1. Implementation Details

Our network is implemented on Pytorch. Except for the Feature Extractor 1, the model (filter weights) are initialized by using the Gaussian distribution with zero-mean and standard deviation 0.01, and the biases are initialized with 0. The hyper-parameters are as follows: mini-batch size=1, learning rate=(1e-2), weight decay=(5e-4), momentum=0.9, and training epochs=120. We divide the learning rate by 10 for every 10 epochs. The SGD solver is used for optimization. All the experiments are conducted on NVIDIA GeForce 2080Ti GPU with 11G memory. The maximum tolerance is 0.0075 for BSDS500 and 0.011 for the NYUDv2 dataset, as defined in the previous works [2, 3, 17].

4.2. Datasets

We train our method using BSDS500 [1], PASCAL VOC [21], and NYUDv2 [22] and evaluate it using BSDS500 and NYUDv2, which has been used by many edge detection researches [2, 3, 16, 17, 18]. Data augmentation has the ability to increase the amount of data available for training the model without actually collecting new data. We implement the data augmentation strategy in [2, 3, 17] by rotating images to 16 different angles, flipping them, and also scaling them.

4.3. Evaluation Metrics

We evaluate our schemes using the popular F-measure [1]. The output of our network is an edge probability map; we need a threshold to produce an edge image. There are two options to set this threshold. First, we set a threshold for all images in the dataset, which is called Optimal Dataset Scale (ODS) [1]. And the second is Optimal Scale Image (OIS) [1], which chooses one threshold for each image.

4.4. Experimental Result and Comparison

We compare our proposed method with several state-of-the-art methods. We categorize the comparison into three. First, a comparison against the traditional approaches. Second, comparison against the machine learning-based approaches, and the last comparison against the deep learning approach. For this comparison, our architecture is trained on the BSDS500 dataset and PASCAL VOC dataset. We evaluate the model using the BSDS500 test set similar to [2, 3, 14, 15, 16, 17, 18, 23]. Table 1 shows the quantitative results against the other edge detection approaches. As we can see, our TIN1 has an extremely small model, at around 80K parameters, which outperforms the traditional methods and most machine learning methods. TIN2 has 0.24M parameters but it achieves an ODS of 0.772, outperforms the BDCN2 [2], which has 0.28M parameters. Fig. 8 shows visual comparisons of our approach and a few others. We also train our architectures on the NYUDv2 [22] training set and validation set. We use Gupta et al. [24] version that split the data set into 381 training, 414 validation and 654 testing images set. We train the NYUDv2 models using the same parameters for BSDS500. Since the NYUDv2 dataset has much fewer data, compared to BSDS500, we increase the epochs to archive a similar number of iterations. For each architecture, TIN1, and TIN2, we train two models: RGB model and HHA representation model. In the evaluation stage, we increase the maximum tolerance to 0.011 as used in [2, 3, 17]. We compare our performance with some other approaches, such as gPb-UCM [1], Structured Forest [12], HED [3], RCF [17], and BDCN [2]. Our models achieve rather good performance compared to the other complicated models. The numerical metrics are shown on Table 2.

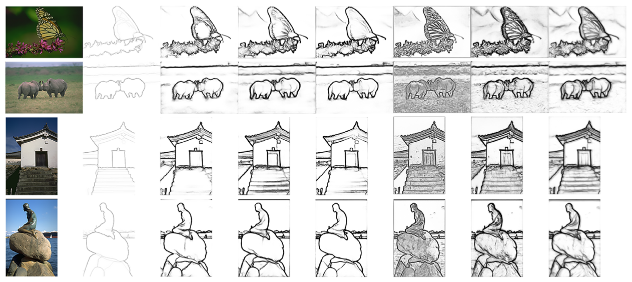

Fig. 8. Edge Detection Comparison on BSDS500. From left to right: input, ground truth, HED [3], RCF[17], BDCN5[2], BDCN2[2], TIN1(Ours), TIN2(Ours)

Table 1. Comparison on BSDS500 test dataset

Method	ODS	IOS	No Params	FPS
Canny [6]	0.611	0.676		28
gPb-UCM [1]	0.729	0.755		1/240
SCG[10]	0.739	0.758		1/18
SF[12]	0.743	0.763		2.5
OEF [13]	0.749	0.772		2/3
DeepEdge [14]	0.753	0.772	-	1/1000
DeepContour[17]	0.757	0.776	-	1/30
HED [3]	0.782	0.804	14.7	30
CEDN [4]	0.788	0.804	119.6	10
RCF [17]	0.806	0.823	14.8	30
RCF-MS [17]	0.811	0.830	14.8	10
CED [16]	0.794	0.811	21.4	30
CED-MS [16]	0.815	0.834	21.4	10
LPCB [18]	0.808	0.824	-	30
LPCB-MS [18]	0.815	0.834	-	10
BDCN2 [2]	0.766	-	0.28	
BDCN3 [2]	0.796	-	2.26	
BDCN4 [2]	0.812	-	8.69	
BDCN [2]	0.820	0.838	16.3	
BDCN-MS [2]	0.828	0.844	16.3	
TIN1 (ours)	0.749	0.772	0.08	30
TIN2 (ours)	0.772	0.795	0.24	30

Table 2. Comparison on BSDS500 test dataset

Method	Model	ODS	IOS
gPb-UCM [1]	RGB	0.632	0.661
OEF [13]	RGB	0.651	0.667
SE [12]	RGB	0.695	0.708
SE+NG+[24]	RGB	0.706	0.734
HED [3]	RGB	0.720	0.734
HED [3]	HHA	0.682	0.695
HED [3]	RGB+HHA	0.746	0.761
RCF [17]	RGB	0.729	0.742
RCF [17]	HHA	0.705	0.715
RCF [17]	RGB+HHA	0.757	0.771
BDCN5 [2]	RGB	0.748	0.763
BDCN5 [2]	HHA	0.707	0.719
BDCN5 [2]	RGB+HHA	0.765	0.781
TIN1 (Ours)	RGB	0.706	0.723
TIN1 (Ours)	HHA	0.661	0.681
TIN1 (Ours)	RGB+HHA	0.729	0.750
TIN2 (Ours)	RGB	0.729	0.745
TIN2 (Ours)	HHA	0.705	0.722
TIN2 (Ours)	RGB+HHA	0.753	0.773

5. CONCLUSIONS

In this paper, we propose a traditional method inspired DNN architecture for edge detection. Different from the previous approaches that borrow the design from the other DNN architectures designed for the other purposes, we develop our system based on the concepts of classical edge detection. We carefully build our model step-by-step and the final system can achieve good edge quality with a much simpler model. Our experiments show that our TIN1 produces good results with an extremely small model. Our TIN2 model has an accuracy higher than the recent BDCN2 but with a smaller model.

6. ACKNOWLEDGEMENT

This work is partially supported by the Ministry of Science and Technology, Taiwan under Grant MOST 109-2634-F-009-020 through Pervasive AI Research (PAIR) Labs, National Chiao Tung University, Taiwan.

7. REFERENCES

[1] Pablo Arbelaez, Michael Maire, Charless C. Fowlkes, and Jitendra Malik, "Contour detection and hierarchical image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 5, pp. 898–916, 2011.

[2] Jianzhong He, Shiliang Zhang, Ming Yang, Yanhu Shan, and Tiejun Huang, "Bi-directional cascade network for perceptual edge detection," in *CVPR*, 2019, pp. 3828–3837.

[3] Saining Xie and Zhuowen Tu, "Holistically-nested edge detection," in *ICCV*, 2015, pp. 1395–1403.

[4] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, Yoshua Bengio and Yann LeCun, Eds., 2015.

[5] Josef Kittler, "On the accuracy of the sobel edge detector," *Image Vis. Comput.*, vol. 1, no. 1, pp. 37–42, 1983.

[6] John F. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, no. 6, pp. 679–698, 1986.

[7] Jitendra Malik, Serge J. Belongie, Thomas K. Leung, and Jianbo Shi, "Contour and texture analysis for image segmentation," *Int. J. Comput. Vis.*, vol. 43, no. 1, pp. 7–27, 2001.

[8] Pedro F. Felzenszwalb and Daniel P. Huttenlocher, "Efficient graph-based image segmentation," *Int. J. Comput. Vis.*, vol. 59, no. 2, pp. 167–181, 2004.

[9] David R. Martin, Charless C. Fowlkes, and Jitendra Malik, "Learning to detect natural image boundaries using local brightness, color, and texture cues," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 5, pp. 530–549, 2004.

[10] Xiaofeng Ren and Liefeng Bo, "Discriminatively trained sparse code gradients for contour detection," in *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012*, Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, Eds., 2012, pp. 593–601.

[11] Joseph J. Lim, C. Lawrence Zitnick, and Piotr Dollár, "Sketch tokens: A learned mid-level representation for contour and object detection," in *CVPR*, 2013, pp. 3158–3165.

[12] Piotr Dollár and C. Lawrence Zitnick, "Fast edge detection using structured forests," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 8, pp. 1558–1570, 2015.

[13] Sam Hallman and Charless C. Fowlkes, "Oriented edge forests for boundary detection," in *CVPR*, 2015, pp. 1732–1740.

[14] Gedas Bertasius, Jianbo Shi, and Lorenzo Torresani, "Deepedge: A multi-scale bifurcated deep network for top-down contour detection," in *CVPR*. 2015, pp. 4380–4389, IEEE Computer Society.

[15] Wei Shen, Xinggang Wang, Yan Wang, Xiang Bai, and Zhijiang Zhang, "Deepcontour: A deep convolutional feature learned by positive-sharing loss for contour detection," in *CVPR*. 2015, pp. 3982–3991, IEEE Computer Society.

[16] Yupei Wang, Xin Zhao, and Kaiqi Huang, "Deep crisp boundaries," in *CVPR*, 2017, pp. 1724–1732.

[17] Yun Liu, Ming-Ming Cheng, Xiaowei Hu, Kai Wang, and Xiang Bai, "Richer convolutional features for edge detection," in *CVPR*, 2017, pp. 5872–5881.

[18] Ruoxi Deng, Chunhua Shen, Shengjun Liu, Huibing Wang, and Xinru Liu, "Learning to predict crisp boundaries," in *ECCV*, 2018, pp. 570–586.

[19] Lee R. Dice, "Measures of the amount of ecologic association between species," *Ecology*, vol. 26, no. 3, pp. 297–302, 1945.

[20] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *CoRR*, vol. abs/1606.00915, 2016.

[21] Roozbeh Mottaghi, Xianjie Chen, Xiaobai Liu, Nam-Gyu Cho, Seong-Whan Lee, Sanja Fidler, Raquel Urtasun, and Alan L. Yuille, "The role of context for object detection and semantic segmentation in the wild," in *CVPR*, 2014, pp. 891–898.

[22] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus, "Indoor segmentation and support inference from RGBD images," in *ECCV*. 2012, vol. 7576 of *Lecture Notes in Computer Science*, pp. 746–760, Springer.

[23] Jimei Yang, Brian L. Price, Scott Cohen, Honglak Lee, and Ming-Hsuan Yang, "Object contour detection with a fully convolutional encoder-decoder network," in *CVPR*, 2016, pp. 193–202.

[24] Saurabh Gupta, Ross B. Girshick, Pablo Andrés Arbeláez, and Jitendra Malik, "Learning rich features from RGB-D images for object detection and segmentation," in *ECCV*, 2014, pp. 345–360.